# Bull

## AIX 5L KDB kernel debugger and kdb command

AIX

ORDER REFERENCE
86 A2 66EM 02

# Bull

## AIX 5L KDB kernel debugger and kdb command

AIX

Software

October 2005

**BULL CEDOC**
**357 AVENUE PATTON**
**B.P.20845**
**49008 ANGERS CEDEX 01**
**FRANCE**

ORDER REFERENCE
86 A2 66EM 02

Printed in France

Suggestions and criticisms concerning the form, content, and presentation of this book are invited. A form is provided at the end of this book for this purpose.

To order additional copies of this book or other Bull Technical Publications, you are invited to use the Ordering Form also provided at the end of this book.

## Trademarks and Acknowledgements

We acknowledge the right of proprietors of trademarks mentioned in this book.

AIX® is a registered trademark of International Business Machines Corporation, and is being used under licence.

UNIX is a registered trademark in the United States of America and other countries licensed exclusively through the Open Group.

Linux is a registered trademark of Linus Torvalds.

# Contents

# About This Book

This book describes how to use the KDB kernel debugger and the kdb command to debug an operating system image. It describes how to examine a stopped kernel in the KDB kernel debugger, as well as how to examine a system dump file using the kdb command. It provides a reference for the commands used to debug the kernel, device drivers, and other kernel extensions for AIX 5L™. Topics include setting breakpoints within the kernel or in kernel extensions, displaying and modifying data structures and instructions, altering system registers, and performing traces. Specific information (for example, syntax and description) is given for each subcommand.

## How to Use This Book

Read the beginning chapters of this book to learn about the KDB kernel debugger and the kdb command.

You can use the alphabetical list to locate a specific subcommand. The subcommand list includes the aliases for each subcommand, a short description of its use, information on when the subcommand can be used and the category to which the subcommand belongs. Subcommands and their aliases are also included in the index.

### Reading Syntax Statements

Syntax statements are a way to represent subcommand syntax and consist of symbols such as brackets ([ ]), braces ({ }), and vertical bars (|). The following is a sample syntax statement:

**examplesubcommand** [ a | b ] [ **-x** *value* ] [ **-y** { address | symbol } ] [ **-z** ] *filename* **...**

The following conventions are used in the command syntax statements:

- Items that must be entered literally on the command line are in bold. These items include the command name, flags, and literal characters.
- Items representing variables that must be replaced by a name are in italics. These items include parameters that follow flags and parameters that the command reads, such as Files and Directories.
- Parameters enclosed in brackets are optional.
- Parameters enclosed in braces are required.
- Parameters not enclosed in either brackets or braces are required.
- A vertical bar signifies that you choose only one parameter. For example, [ a | b ] indicates that you can choose a, b, or nothing. Similarly, { a | b } indicates that you must choose either a or b.
- Ellipses ( ... ) signify the parameter can be repeated on the command line.
- The dash ( - ) represents standard input.

## Highlighting

The following highlighting conventions are used in this book:

| | |
|---|---|
| **Bold** | Identifies commands, subroutines, keywords, files, structures, directories, and other items whose names are predefined by the system. Also identifies graphical objects such as buttons, labels, and icons that the user selects. |
| *Italics* | Identifies parameters whose actual names or values are to be supplied by the user. |
| `Monospace` | Identifies examples of specific data values, examples of text similar to what you might see displayed, examples of portions of program code similar to what you might write as a programmer, messages from the system, or information you should actually type. |

## Case-Sensitivity in AIX

Everything in the AIX® operating system is case-sensitive, which means that it distinguishes between uppercase and lowercase letters. For example, you can use the **ls** command to list files. If you type `LS`, the system responds that the command is ″not found.″ Likewise, **FILEA**, **FiLea**, and **filea** are three distinct file names, even if they reside in the same directory. To avoid causing undesirable actions to be performed, always ensure that you use the correct case.

## ISO 9000

ISO 9000 registered quality systems were used in the development and manufacturing of this product.

## Related Publications

The following books contain information about or related to debugging programs:

- *AIX 5L Version 5.3 Kernel Extensions and Device Support Programming Concepts*

# Chapter 1. KDB kernel debugger and kdb command

This document describes the KDB kernel debugger and **kdb** command. The KDB kernel debugger and the **kdb** command are the primary tools a developer uses for debugging device drivers, kernel extensions, and the kernel itself. Although they appear similar to the user, the KDB kernel debugger and the **kdb** command are two separate tools:

**KDB kernel debugger**
: The KDB kernel debugger is integrated into the kernel and allows full control of the system while a debugging session is in progress. The KDB kernel debugger allows for traditional debugging tasks such as setting breakpoints and single-stepping through code.

**kdb command**
: This command is implemented as an ordinary user-space program and is typically used for post-mortem analysis of a previously-crashed system by using a system dump file. The **kdb** command includes subcommands specific to the manipulation of system dumps.

Both the KDB kernel debugger and **kdb** command allow the developer to display various structures normally found in the kernel's memory space. Both do the following:

- Provide numerous subcommands to decode various data structures found throughout the kernel.
- Print the data structures in a user-friendly format.
- Perform debugging at the machine instruction level. Although this is less convenient than source level debugging, it allows the KDB kernel debugger and the **kdb** command to be used in the field where access to source code might not be possible.
- Process the debugging information found in XCOFF objects. This allows the use of symbolic names for functions and global variables.

The following sections describe more about the KDB kernel debugger and **kdb** command:


- "KDB kernel debugger"
- "The kdb command" on page 5


The following sections outline how to invoke the KDB kernel debugger and **kdb** command:


- "Invoking the KDB kernel debugger" on page 2
- "Invoking the kdb command" on page 5


## KDB kernel debugger

Although it must be manually enabled by the user prior to use, the KDB kernel debugger is statically compiled into the AIX kernel and is always loaded. After it is enabled, the KDB kernel debugger can be manually invoked by the user or automatically invoked by the system in response to some condition (for example, an unhandled exception in the kernel code). For more information, see "Invoking the KDB kernel debugger" on page 2.

KDB kernel debugger is always loaded into a special region of pinned memory where the effective address space equals the real address space. The KDB kernel debugger runs with memory translation turned off. This allows it to function even if the VMM subsystem is not yet initialized or the critical VMM structures are corrupted. However, the KDB kernel debugger can perform the same address translations normally performed by the processor. This allows the user to view data by effective addresses when the processor has its memory translation turned off.

When the KDB kernel debugger is invoked by a condition, it is the only running program. All other processes are stopped and processor interrupts are disabled. One of the processors is designated as the debug processor and that processor runs the KDB kernel debugger. This is usually the processor on which an unusual activity occurred (for example, an unhandled exception).

If the KDB kernel debugger is invoked manually by the user, the debug processor is arbitrarily chosen. The KDB kernel debugger stops all other processors in the system by sending an interprocessor interrupt (IPI) to each processor. If any of these processors cannot be stopped, the KDB kernel debugger prints a warning message. For example, if a processor is spinning on a lock with interrupts disabled, it cannot process the IPI sent by the KDB kernel debugger.

The KDB kernel debugger is mostly self-contained and does not rely on other kernel components such as the network and video drivers. The KDB kernel debugger runs with its own Machine State Save Area (mst) and a special stack. This requires that some kernel code be duplicated within KDB kernel debugger. Duplication allows the developer to debug from almost anywhere within the kernel code. Unless the KDB kernel debugger is entered through a system halt, processors resume normal operation and interrupts are re-enabled when the developer exits the KDB kernel debugger.

When it is invoked, the KDB kernel debugger takes control of either the virtual terminal (vterm) on a logical partitioning system, or a physical RS232 serial port on a non-logical partitioning system. This requires a Hardware Management Console (HMC) to access the vterm or another system connected to the serial port on the system being debugged. The KDB kernel debugger requires the connection in order to send messages to the developer.

The complete list of subcommands available for the KDB kernel debugger and **kdb** command are included in Chapter 7, “Subcommand lists,” on page 29.

## Invoking the KDB kernel debugger

This topic describes how to load and start the KDB kernel debugger, and what you need to know about terminal use. For information on how to invoke the **kdb** command, see “Invoking the kdb command” on page 5.

### Loading and starting the KDB kernel debugger in AIX 5.1 and subsequent releases

For AIX 5.1 and subsequent releases, the KDB kernel debugger is the standard kernel debugger and is included in the unix_up and unix_mp kernels, which are in the **/usr/lib/boot** file.

The KDB kernel debugger must be loaded at boot time. This requires that a boot image be created with the debugger enabled. To enable the KDB kernel debugger, use either the *-I* or *-D* options of the **bosboot** command.

Examples of **bosboot** commands are as follows:

- To disable the KDB kernel debugger, use the following command:
  ```
  bosboot -a -d /dev/ipldevice
  ```
- To enable the KDB kernel debugger, but not invoke it during system initialization, use the following command:
  ```
  bosboot -a -d /dev/ipldevice -D
  ```
- To enable the KDB kernel debugger, and invoke it during system initialization, use the following command:
  ```
  bosboot -a -d /dev/ipldevice -I
  ```

**Notes:**

1. **bosboot** commands build boot images using the KDB kernel debugger. The boot image is not used until the machine is restarted.
2. External interrupts are disabled while the KDB kernel debugger is active.
3. If invoked during system initialization, the **g** subcommand must be issued to continue the initialization process.

For more information on the **bosboot** command, see *AIX 5L Version 5.3 Commands Reference, Volume 1*

## Loading and starting the KDB kernel debugger in AIX 4.3.3

The KDB kernel debugger must be loaded at boot time. This requires that a boot image be created with the debugger enabled. To enable the KDB kernel debugger, the **bosboot** command must be invoked with a KDB kernel specified and options set to enable the KDB kernel debugger. KDB kernels are shipped as **/usr/lib/boot/unix_kdb** for uni-processor (UP) systems and **/usr/lib/boot/unix_mp_kdb** for Multi-processor (MP) systems. The specific kernel used to create the boot image can be specified using the *-k* option of the **bosboot** command. The KDB kernel debugger must also be enabled using either the *-I* or *-D* options of the **bosboot** command.

Examples of **bosboot** commands for a UP system are as follows:

- To disable the KDB kernel debugger, use the following command:
  ```
  bosboot -a -d /dev/ipldevice -k /usr/lib/boot/unix_kdb
  ```
- To enable the KDB kernel debugger, but not invoke it during system initialization, use the following command:
  ```
  bosboot -a -d /dev/ipldevice -D -k /usr/lib/boot/unix_kdb
  ```
- To enable the KDB kernel debugger, and invoke it during system initialization, use the following command:
  ```
  bosboot -a -d /dev/ipldevice -I -k /usr/lib/boot/unix_kdb
  ```

**Notes:**

1. For an MP system, the **/usr/lib/boot/unix_mp_kdb** file is used instead of the **/usr/lib/boot/unix_kdb** file.
2. The **bosboot** commands build boot images using the KDB kernel debugger. The boot image is not used until the machine is restarted.
3. External interrupts are disabled while the KDB kernel debugger is active.
4. If invoked during system initialization, the **g** subcommand must be issued to continue the initialization process.

For more information about the **bosboot** command, see *AIX 5L Version 5.3 Commands Reference, Volume 1*

The **/usr/lib/boot/unix** and **/unix** links are not changed by the **bosboot** command. However, these links are used by user commands such as **sar** and others to read symbol information for the kernel. If these commands are to be used with a KDB boot image `/unix` and `/usr/lib/boot/unix` must point to the kernel specified for the **bosboot** command. This can be done by removing and recreating the links. This must be done as the root user. For the previous **bosboot** command examples, typing the following would set up the links correctly:

1. Type
   ```
   rm /unix
   ```
   and press Enter.
2. Type
   ```
   ln -s /usr/lib/boot/unix_kdb /unix
   ```
   and press Enter.
3. Type
   ```
   rm /usr/lib/boot/unix
   ```
   and press Enter.
4. Type
   ```
   ln -s /usr/lib/boot/unix_kdb /usr/lib/boot/unix
   ```
   and press Enter.

Similarly, if you chose to stop using a KDB Kernel, the links for **/unix** and **/usr/lib/boot/unix** should be modified to point to the kernel specified to the **bosboot** command.

**Note:** `/unix` is the default kernel used by the **bosboot** command. If this link is changed to point to a KDB kernel, after **bosboot** commands that do not have a kernel specified are run, the commands use the KDB kernel.

## Entering the KDB kernel debugger

Enter the KDB kernel debugger using one of the following procedures:

- On a tty keyboard, press the Ctrl+4 key sequence for IBM® 3151 terminals or the Ctrl+\ key sequence for BQ 303, BQ 310C, and WYSE 50 terminals.
- On other keyboards, press the Ctrl+Alt+Numpad4 key sequence.
- Set a breakpoint using one of the Chapter 17, “Breakpoint and steps subcommands,” on page 115.
- Call the **brkpoint** subroutine from the C code. The syntax for calling this subroutine is the following:

```
brkpoint();
```

**Note:** The system enters the debugger if a system halt is caused by a fatal system error. In such a case, the system creates a log entry in the system log and if the KDB kernel debugger is available, it is called. A system dump might be generated when you exit from the debugger.

If the kernel debug program is not available when you type in a key sequence, you must load the kernel debug program.

For more information about loading the kernel debug program, see “Loading and starting the KDB kernel debugger in AIX 4.3.3” on page 3 or “Loading and starting the KDB kernel debugger in AIX 5.1 and subsequent releases” on page 2.

You can use the **kdb** command with the **dw** subcommand to determine whether the KDB kernel debugger is available by typing the following:

```
# kdb
(0)> dw kdb_avail
(0)> dw kdb_wanted
```

**Note:** If either of the previous **dw** subcommands returns a `0`, the KDB kernel debugger is not available.

After the KDB kernel debugger is invoked, the subcommands detailed in Chapter 7, “Subcommand lists,” on page 29 are available.

## Using a terminal with the KDB kernel debugger

**Note:** If you are using the Hardware Management Console, KDB kernel debugger can be accessed using a virtual terminal. For more information, see the *Hardware Management Console Installation and Operations Guide* (SA38 – 0590).

The KDB kernel debugger opens an asynchronous ASCII terminal when it is first started, and subsequently upon being started due to a system halt. Native serial ports are checked sequentially, starting with port 0 (zero). Each port is configured at 9600 bps, 8 bits, and no parity. If carrier detect is asserted within 1/10 of a second, the port is used. Otherwise, the next available native port is checked. This process continues until a port is opened or until every native port available on the machine is checked. If no native serial port is opened successfully, the result is unpredictable.

The KDB kernel debugger only supports display to an ASCII terminal connected to a native serial port. Displays connected to graphics adapters are *not* supported. The KDB kernel debugger uses its own device driver for handling the display terminal. It is possible to connect a serial line between two machines and define the serial line port as the port for the console. In that case, the **cu** command can be used to connect to the target machine and run the KDB kernel debugger.

**Note:** If a serial device, other than a terminal connected to a native serial port, is selected by the kernel debugger, the system might appear to hang.

## The kdb command

The **kdb** command can be used for analyzing the following:

- A running system.

  When used to analyze a running system, the **kdb** command opens the **/dev/pmem** special file, which allows direct access to the system's physical memory and bypasses the normal address translation mechanism of the processor. The **kdb** command performs its own address translation internally using the same algorithms as the KDB kernel debugger. This allows the user to view data by effective address.

  **Note:** Only the root user can use the **kdb** command to analyze a running system.
- A system dump file produced by a previously crashed-system.

  When a system crashes, the system dump image is created with memory translation turned on. As a result, any physical memory not mapped to the effective address space at the time of the dump cannot be included in the dump file. Only the memory belonging to the process that was running on the processor that created the dump image can be included in the dump file. Because all addresses within the system dump are already effective addresses, the **kdb** command does not perform its internal address translation.

  A system dump contains certain critical data structures. A system dump does not contain the entire effective address space. The **kdb** command might not be able to view certain memory regions. If someone attempts to access a memory address not included in the dump, the **kdb** command prints a warning message.

  **Note:** The **cdt** subcommand or the *-v* command-line option can be used to determine exactly which regions of the effective address space are included in the system image. For more information about the **CDT** subcommand, see “cdt subcommand” on page 384. For more information about the *-v* command line option, see Appendix A, “kdb Command,” on page 429.

The **kdb** command contains a subset of the subcommands found in the KDB kernel debugger. Subcommands for setting breakpoints and single-stepping through code are not available in the **kdb** command. Because the **kdb** command is implemented as an ordinary user-space program, it has no control over the processors in a system. Similarly, any subcommands that directly access hardware (for example, the PCI subcommands) are not available. When you work with a system dump, any subcommands that modify memory are not valid because the system dump is merely a snapshot of the real memory in a system.

The complete list of subcommands available for the KDB kernel debugger and **kdb** command are included in Chapter 7, “Subcommand lists,” on page 29.

## Invoking the kdb command

This topic describes how to configure a processor for system dumps, obtain and verify a system dump, and run the **kdb** command. To analyze a running system, the **kdb** command is simply invoked from the UNIX® shell prompt without any command line arguments.

**Note:** Because the **kdb** command makes use of the **/dev/pmem** special file when analyzing a running system, only the root user can invoke the command in this manner.

A side effect of analyzing the running system with the **kdb** command is that the currently running process as displayed with the **p *** subcommand, often appears to be the **kdb** command itself. This occurs because the **kdb** command can only read the **/dev/pmem** special file when it is the current process on one of the processors in the system.

When you are analyzing a system dump file, the **kdb** command must be started with command line arguments that specify the location of the dump files and the kernel files as shown in the following example:

```
# kdb /var/adm/ras/vmcore.0 /unix
```

The kernel file is used by the **kdb** command to resolve symbol names from the dump file. It is imperative that the kernel file specified on the command line is the kernel file that was running at the time the system dump was created.

For more information about creating system dumps, see System Dump Facility in *AIX 5L Version 5.3 Kernel Extensions and Device Support Programming Concepts*.

For more information about invoking the KDB kernel debugger, see “Invoking the KDB kernel debugger” on page 2.

# Chapter 2. The debugger prompt

All work in the KDB kernel debugger and the **kdb** command is performed at the debugger prompt. On a uniprocessor system, the KDB kernel debugger prompt is `KDB(0)>` and the **kdb** command prompt is `(0)>`. When you are debugging a multiprocessor system, the number enclosed in parentheses indicates the processor that is being debugged. Many subcommands, such as those that display or modify registers, apply only to the current processor.

As shown in the following example, the **cpu** subcommand can be used to change the current processor:

```
(0)> dr r1
r1  : 2FF3B338   2FF3B338
(0)> cpu 1
(1)> dr r1
r1  : 2FF3AA20   2FF3AA20
(1)>
```

Many subcommands can produce a large amount of output. To keep the output from scrolling off the screen, the debugger implements a pager which displays a more `(^C to quit) ?` prompt after each full screen of data. When you see the prompt, you can do one of the following:

- Press the space bar to view the next line of output.
- Press the Enter key to view the next page of output.
- Press Ctrl+C to abort the current subcommand and return to the main debugger prompt.

The pager is controlled with the **set** subcommand using the *screen_size* and *scroll* options. For more information, see the "set subcommand" on page 44.

## Online help

The **help** subcommand can be typed at any time to display a list of all available subcommands and a one-line description of each of the subcommands. Many subcommands also allow a `-?` parameter that displays a more detailed description of that subcommand. For example, to see a list of display context subcommands, type the following at the command prompt:

```
help display context
```

The following results are displayed:

```
CMD      ALIAS    ALIAS    FUNCTION                   ARG

                *** display context information ***

pnda                       Display pnd area           [*][-a][cpunb/symb/eaddr]
ppda                       Display ppd area           [*/cpunb/symb/eaddr]
mst                        Display mst area           [slot] [[-a] symb/eaddr]
lastbackt                   Display lastbackt          cpu number
p        proc              Display proc table         [*/slot/symb/eaddr]
th       thread            Display thread table       [*/slot/symb/eaddr/-w ?]
ttid     th_tid            Display thread tid         [tid]
tpid     th_pid            Display thread pid         [pid]
rq       runq              Display run queues         [bucket/symb/eaddr]
rqi      rqa               Display RQ Info
sq       sleepq            Display sleep queues       [bucket/symb/eaddr]
lq       lockq             Display lock queues        [bucket/symb/eaddr]
u        user              Display u_area             [-?][slot/symb/eaddr]
cr       crid              Display crid table         [*/slot/symb/eaddr]
chkfile                    Display chkfile structure  eaddr
svmon                      Process based paging space and mem usage [-?]
```

For example, to see a list of parameters for the **p** subcommand and a brief description of what the parameter does, type the following at the command prompt:

```
p -?
```

The following results are displayed:

```
PROC USAGE:'p ?' print usage
PROC USAGE:'p' print current process
PROC USAGE:'p *' print process table
PROC USAGE: 'p -' print all processes in none/zombie state in long format
PROC USAGE: 'p <slot>' print process in <slot>
PROC USAGE: 'p <address>' print process at <address>
PROC USAGE: 'p <symbol>' print process matching <symbol>
PROC USAGE: 'p -s <proc state>'sort processes by state
PROC USAGE: 'p -n <substring>'sort processes by name
(0)>
```

For an alphabetic list of the subcommands, see Chapter 7, "Subcommand lists," on page 29. Because the -? parameter is available with most subcommands, this parameter is not included in the detailed subcommand descriptions found in this book.

## Registers

Register values can be referenced by the KDB kernel debugger and the **kdb** command. Register values can be used in subcommands by preceding the register name with an at sign (@). This character is also used to dereference addresses as described in "Expressions" on page 10. Registers that can be referenced include the following:

| Register | Description |
|---|---|
| asr | Address space register |
| cr | Condition register |
| ctr | Count register |
| dar | Data address register |
| dec | Decrementer |
| dsisr | Data storage interrupt status register |
| fp0-fp31 | Floating point registers 0 through 31 |
| fpscr | Floating point status and control register |
| iar | Instruction address register |
| lr | Link register |
| mq | Multiply quotient |
| msr | Machine State register |
| r0-r31 | General Purpose Registers 0 through 31 |
| rtcl | Real Time clock (nanoseconds) |
| rtcu | Real Time clock (seconds) |
| s0-s15 | Segment registers |
| sdr0 | Storage description register 0 |
| sdr1 | Storage description register 1 |
| srr0 | Machine status save/restore 0 |
| srr1 | Machine status save/restore 1 |
| tbl | Time base register, lower |
| tbu | Time base register, upper |
| tid | Transaction register (fixed point) |

| Register | Description |
| --- | --- |
| xer | Exception register (fixed point) |

Other special purpose registers that can be referenced, if they are supported on the hardware, include the following:

- sprg0
- sprg1
- sprg2
- sprg3
- pir
- fpecr
- ear
- pvr
- hid0
- hid1
- iabr
- dmiss
- imiss
- dcmp
- icmp
- hash1
- hash2
- rpa
- buscsr
- l2cr
- l2sr
- mmcr0
- mmcr1
- pmc1
- pmc2
- pmc3
- pmc4
- pmc5
- pmc6
- pmc7
- pmc8
- sia
- sda

## Expressions

The KDB kernel debugger and **kdb** command can parse a limited set of expressions. Expressions can only contain symbols, hexadecimal constants, references to register or memory locations, and operators. Supported operators include the following:

| **Operator** | **Definition** |
|---|---|
| + | Addition |
| - | Subtraction |
| * | Multiplication |
| / | Division |
| % | Modulo |
| ^ | Exponentiation |
| ( ) | Parenthesis (order of operations) |
| @ | Dereferencing |

The dereference operator does the following:

- Indicates that the value at the location indicated by the next operand is to be used in the calculation of the expression.

  For example, `@f000` indicates that the value at address `0x0000f000` should be used in evaluation of the expression.
- Allows access to the contents of a register.

  For example, `@r1` references the contents of general purpose register 1. Recursive dereferencing is allowed. As an example, `@@r1` references the value at the address pointed to by the value at the address contained in general purpose register 1.

The + and - operators have equal precedence. Likewise, the * / % and ^ operators have equal precedence with each other. Multiple operators with the same precedence are always evaluated from left to right in an expression. The following are examples:

| **Valid Expressions** | **Results** |
|---|---|
| `dw @r1` | Displays data at the location pointed to by r1. |
| `dw @@r1` | Displays data at the location pointed to by value at location pointed to by r1. |
| `dw open` | Displays data at the address beginning of the open routine. |
| `dw open+12` | Displays data twelve bytes past the beginning of the open routine. |
| **Invalid Expressions** | **Problem** |
| `dw r1` | Must include the at sign (@) to reference the contents of r1, If a symbol r1 existed, this would be valid. |

## User-defined variables

Both the KDB kernel debugger and the **kdb** command allow for user-defined variables. These variables can be used to provide a custom name for a memory address or an alias for a commonly used subcommand. After a user-defined variable is created, every occurrence of that variable in a subcommand is automatically replaced with the value assigned to the variable.

Variable substitution occurs before any other parsing of the subcommand. This allows a single variable to expand into multiple subcommand arguments. The **varset**, **varrm**, and **varlist** subcommands are used respectively for assigning, removing, and listing user-defined variables. The following is an example of how user-defined variables are used:

```
KDB(0)> varset myvar kdb_avail
KDB(0)> dw myvar
<<dw kdb_avail>>
kdb_avail+000000: 00000001 00000000 0800004C 00001C43  ...........L...C
KDB(0)> varset myvar kdb_avail 1
KDB(0)> dw myvar
<<dw kdb_avail 1>>
kdb_avail+000000: 00000001                                      ....
KDB(0)>
```

Any time a user variable expansion takes place at the debugger prompt, the expanded command line is printed between the << and >> marks.

## Command line editing

Command line editing at the `KDB(0)>` or `(0)>` debugger prompt is supported and includes a history of recent commands. In addition, the command line supports several emacs and vi key bindings for editing text.

The **set** subcommand can be used to select the edit mode. The edit mode determines the set of key bindings that is currently active.

Regardless of which editing mode is used, the Ctrl+S and the Ctrl+Q key sequences are always available. The Ctrl+S key sequence pauses the debugger's output to the screen and the Ctrl+Q key sequence causes the output to continue to resume the screen display.

### The emacs or gmacs editing mode

If the emacs or gmacs mode is active, the following key bindings are supported:

| Key Sequence | Associated Action |
|---|---|
| Ctrl+F | Move the cursor one character forward. |
| Ctrl+B | Move the cursor one character backward. |
| Ctrl+A | Move the cursor to the beginning of the command line. |
| Ctrl+E | Move the cursor to the end of the command line. |
| Ctrl+P | Display the previous command in the history buffer. |
| Ctrl+N | Display the next command in the history buffer. |
| Ctrl+D | Delete the character at the cursor position. |
| Ctrl+U | Delete the entire command line. |
| Ctrl+T | In emacs mode, transpose the current and previous characters. In gmacs mode, transpose the previous two characters. |

In addition the emacs and gmacs modes, allow a repeat count to be used with several of the above key sequences. If the Esc key is pressed followed by one or more numbers, and finally one of the above Ctrl key sequences is pressed, then the numbers following the Esc key are interpreted as a repeat count for the final Ctrl key sequence.

## The vi editing mode

When the vi edit mode is active, the command prompt can be in either the vi text-input mode or the **vi** command mode. The command line starts in text-input mode where all typed characters become part of the text on the command line. Pressing the Esc key while in text-input mode switches your screen to the **vi** command mode. In the command mode, the debugger recognizes the following standard **vi** subcommands: `l w W e E h b B | ^ $ f F t T ; , k - j + G ? / n N . a i A s S R ~ I C D x X p P Y r y d c u` and `U`.

**Note:** Any **vi** subcommands that begin with a colon are not supported.

For more information about **vi** subcommands, see vi subcommands in *AIX 5L Version 5.3 Commands Reference, Volume 6*.

## Multiprocessor systems

On multiprocessor systems, entering the KDB kernel debugger stops all processors except the current processor running the debug program itself. On multiprocessor systems, the number in parentheses that is part of the prompt indicates the current processor. For example:

- For the following prompt, `KDB(0)>`, the number `0` is contained in parentheses and is the current processor.
- For the following prompt, `KDB(5)>`, the number `5` is contained in parentheses and is the current processor.

In addition to the change in the prompt for multiprocessor systems, there are also subcommands that are unique to these systems. For more information about the subcommands that can be used on multiprocessor systems, see Chapter 7, “Subcommand lists,” on page 29. The subcommands that are unique to multiprocessors are identified in the usage column.

# Chapter 3. Viewing and modifying global data

**Note:** The **demo** and **demokext** programs are used in the examples in this section. The *demokext_j* variable, which is exported is used in the examples.

Global data can be accessed using several methods:

- "Method 1: Using the symbol name" demonstrates the simplest method of accessing global data. This is the primary method of accessing global data when using the KDB kernel debugger. The other methods are described to show alternatives and to allow the use of additional KDB subcommands in examples.
- "Method 2: Using the TOC and map file" on page 14 demonstrates accessing global data using the TOC and the map file. This method requires that the system is stopped in the KDB kernel debugger within a procedure of the kernel extension to be debugged. The address of the data for the *demokext_j* variable is calculated.
- "Method 3: Using the map file" on page 15 demonstrates a way to access global data using the map file, but without using the TOC. The address of the data for the *demokext_j* variable is calculated.

Before using any of the following examples, see "Loading the kernel extension" on page 431.

## Method 1: Using the symbol name

Global variables within the KDB kernel debugger can be accessed directly by name. For example, the **dw** subcommand can be used to display the value of the *demokext_j* variable. If the *demokext_j* variable is an array, a specific value can be viewed by adding the appropriate offset (for example, `dw demokext_j+20`). Access to individual elements of a structure is accomplished by adding the proper offset to the base address for the variable.

**Note:** The default prompt is `KDB(0)>`.

To view and modify global variables using the symbol name, do the following:

1. Display a word at the address of the *demokext_j* variable with the following command:
   ```
   dw demokext_j
   ```
   Because the kernel extension was just loaded, this variable should have a value of `99` and the KDB kernel debugger should display that value. The data displayed should be similar to the following:
   ```
   demokext_j+000000: 00000063 01304040 01304754 00000000  ...c.0@@.0GT....
   ```
2. Turn off symbolic name translation by typing the following:
   ```
   ns
   ```
3. To display the word at the address of the *demokext_j* variable, type the following:
   ```
   dw demokext_j
   ```
   With symbolic name translation turned off, the data displayed should be similar to the following:
   ```
   01304744: 00000063 01304040 01304754 00000000  ...c.0@@.0GT....
   ```
4. Turn symbolic name translation on by typing the following:
   ```
   ns
   ```
5. Modify the word at the address of the *demokext_j* variable by typing the following:
   ```
   mw demokext_j
   ```
   The KDB kernel debugger displays the current value of the word and waits for user input to change the value. The data displayed should be similar to the following:
   ```
   01304744:  00000063  =
   ```

Type a new value and press Enter. After a new value is entered, the next word of memory is displayed for possible modification. To end memory modification type a period (.) and press Enter. Type a value of 64 (100 decimal) for the first address, type a period and press Enter to end modification.

## Method 2: Using the TOC and map file

Before you can locate the address of global data using the address of the TOC and the map file, the system must be stopped in the KDB kernel debugger within a routine of the kernel extension you want to debug. To do this, set a breakpoint within the kernel extension. For more information about setting a breakpoint, see Chapter 5, "Setting breakpoints," on page 21.

When the KDB kernel debugger is invoked, general purpose register number 2 points to the address of the TOC. From the map file, the offset from the start of the table of contents (TOC) to the desired TOC entry can be calculated. Knowing this offset, and knowing the address at which the TOC starts, allows the address of the TOC entry for the desired global variable to be calculated. Then, the address of the TOC entry for the desired variable can be examined to determine the address of the data.

For example, assume that the KDB kernel debugger was invoked because of a breakpoint at line 67 of the **demokext** routine, and that the value for general purpose register number 2 is `0x01304754`.

To find the address of the *demokext_j* variable, complete the following:

1. Calculate the offset from the beginning of the TOC to the TOC entry for the *demokext_j* variable. From the map file, the TOC starts at `0x0000010C` and the TOC entry for the *demokext_j* variable is at *0x00000114*. Therefore, the offset from the beginning of the TOC to the entry of interest is:
   ```
   0x00000114 - 0x0000010C = 0x00000008
   ```
2. Calculate the address of the TOC entry for the *demokext_j* variable. This is the current value of general purpose register 2 plus the offset calculated in the preceding step. The calculation is as follows:
   ```
   0x01304754 + 0x00000008 = 0x0130475C
   ```
3. Display the data at `0x0130475C`. The data displayed is the address of the data for *demokext_j*.

To view and modify global data, do the following:

1. At the `KDB(0)` prompt, set a break at line 67 of the **demokext** routine by typing the following:
   ```
   b demokext+e0
   ```
   **Note:** Breaking at this location ensures that the KDB kernel debugger is invoked while within the **demokext** routines.
2. Obtain the value of General Purpose Register 2. You need that to determine the address of the TOC.
3. Exit the KDB kernel debugger by typing `g` on the command line.
4. Bring the demo program to the foreground and choose a selection. Choosing a selection causes the **demokext** routine to be called for configuration. Because a break was set, this causes the KDB kernel debugger to be invoked.

   **Note:** The prompt changes to a dollar sign (`$`).
5. Bring the demo program to the foreground by typing the following:
   ```
   fg
   ```
   **Note:** The prompt changes to `./demo`.
6. Enter a value of `1` to select the option to increment the counters within the **demokext** kernel extension. This causes a break at line 67 of the **demokext** kernel extension and the prompt changes to `KDB(0)`.
7. Display the general purpose registers by typing the following:
   ```
   dr
   ```

The data displayed should be similar to the following:

```
r0  : 0130411C  r1  : 2FF3B210  r2  : 01304754  r3  : 01304744  r4  : 0047B180
r5  : 0047B230  r6  : 000005FB  r7  : 000DD300  r8  : 000005FB  r9  : 000DD300
r10 : 00000000  r11 : 00000000  r12 : 013042F4  r13 : DEADBEEF  r14 : 00000001
r15 : 2FF22D80  r16 : 2FF22D88  r17 : 00000000  r18 : DEADBEEF  r19 : DEADBEEF
r20 : DEADBEEF  r21 : DEADBEEF  r22 : DEADBEEF  r23 : DEADBEEF  r24 : 2FF3B6E0
r25 : 2FF3B400  r26 : 10000574  r27 : 22222484  r28 : E3001E30  r29 : E6001800
r30 : 01304744  r31 : 01304648
```

Using the map, the offset to the TOC entry for the *demokext_j* variable from the start of the TOC was `0x00000008`. Adding this offset to the value displayed for r2 indicates that the TOC entry of interest is at: `0x0130475C`.

**Note:** The KDB kernel debugger can be used to perform the addition. In this case, the subcommand to use is **hcal** *@r2+8*. For more information about the **hcal** subcommand, see “hcal and dcal subcommands” on page 70.

8. Display the TOC entry for the *demokext_j* variable by typing the following:

```
dw 0130475C
```

This entry contains the address of the data for the *demokext_j* variable. The data displayed should be similar to the following:

```
TOC+000008: 01304744 000BCB34 00242E94 001E0518  .0GD...4.$......
```

The value for the first word displayed is the address of the data for the *demokext_j* variable.

9. Display the data for the *demokext_j* variable by typing the following:

```
dw 01304744
```

The displayed data should indicate that the value for the *demokext_j* variable is still 0x0000064. This was set earlier because the breakpoint set was in the **demokext** routine prior to incrementing the *demokext_j* variable. The data displayed should be similar to the following:

```
demokext_j+000000: 00000064 01304040 01304754 00000000  ...d.0@@.0GT....
```

10. Clear all breakpoints with the following command:

```
ca
```

11. Exit the kernel debugger by typing `g` on the command line.

**Note:** When you exit, the demo program is in the foreground and a prompt for the next option is displayed. The kernel extension is going to run and increment the *demokext_j* variable. Next time it should have a value of `0x00000065`.

12. Type the Ctrl+Z key sequence to stop the demo program. At this point, the prompt changes to a dollar sign ($).
13. Place the demo program in the background by typing the following:

```
bg
```

## Method 3: Using the map file

Unlike the procedure outlined in “Method 2: Using the TOC and map file” on page 14, this method can be used at any time. This method requires the map file and the address at which the kernel extension was loaded.

**Note:** Because this method depends on how a kernel extension is loaded, this method might quit working if the procedure for loading a kernel extension is changed.

This method relies on the assumption that the address of a global variable can be found by using the following formula:

```
Addr of variable = Addr of the last function before the variable in the map +
                   Length of the function +
                   Offset of the variable
```

The following is a part of the map file for the **demokext** kernel extension:

```
20       000005B8 000028  2 GL SD S17   <.fp_write>              glink.s(/usr/lib/glink.o)
21       000005B8           GL LD S18   .fp_write
22       000005E0 000028  2 GL SD S19   <.fp_open>               glink.s(/usr/lib/glink.o)
23       000005E0           GL LD S20   .fp_open
24       00000000 0000F9  3 RW SD S21   <_$STATIC>               demokext.c(demokext.o)
25     E 000000FC 000004  2 RW SD S22   demokext_j               demokext.c(demokext.o)
26   *   00000100 00000C  2 DS SD S23   demokext                 demokext.c(demokext.o)
27       0000010C 000000  2 T0 SD S24   <TOC>
28       0000010C 000004  2 TC SD S25   <_$STATIC>
29       00000110 000004  2 TC SD S26   <_system_configuration>
```

The last function in the **.text** section is at lines 22 and 23. The offset of this function from the map is `0x000005E0` (line 22, column 2). The length of the function is `0x000028` (Line 22, column 3). The offset of the *demokext_j* variable is `0x000000FC` (line 25, column 2). So the offset from the load point value to the *demokext_j* variable is:

```
0x000005E0 + 0x000028 + 0x000000FC = 0x00000704
```

Adding this offset to the load point value of the **demokext** kernel extension provides the address of the data for the *demokext_j* variable. Assuming a load point value of `0x01304040`, this indicates that the data for the *demokext_j* variable is located at:

```
0x01304040 + 0x00000704 = 0x01304744
```

To view global data, complete the following:

1. Activate KDB kernel debugger. Use the appropriate key sequence for your configuration. When this step is complete, you should see a KDB prompt.
2. Display the data for the *demokext_j* variable by typing the following:

   ```
   dw demokext+704
   ```

   The `704` value is calculated from the map using the procedure listed above. This offset is then added to the load point of the **demokext** routine. The value for the *demokext_j* variable should now be 0x00000065. The data displayed should be similar to the following:

   ```
   demokext_j+000000: 00000065 01304040 01304754 00000000  ...e.0@@.0GT....
   ```

   **Note:** There are numerous ways to find this address. For other methods, see Chapter 5, “Setting breakpoints,” on page 21.
3. Exit the KDB kernel debugger by typing `g` on the command line and pressing Enter. The prompt changes to a dollar sign ($).
4. Bring the demo program to the foreground by typing`fg` and pressing Enter. The prompt changes to `./demo`.
5. Type `0` and press Enter to unload the **demokext** kernel extension and exit.

# Chapter 4. Viewing stack traces

This topic describes:

- “Stack frame format”
- “Verbose stack output” on page 19

**Note:** The examples in this topic assume that the current process is the demonstration program that called the **demokext** kernel extension because there was a breakpoint set.

## Stack frame format

To learn how to view and manipulate stack frame formats, perform the following steps:

1. Load the **demokext** kernel extension program. For directions, see “Loading the kernel extension” on page 431.
2. Display the stack for the current process, by typing `stack` and pressing Enter.

   The stack trace back displays the routines called and traces back through system calls. The displayed data should be similar to the following:

```
thread+001800 STACK:
[013042C0]write_log+00001C (10002040, 2FF3B258, 2FF3B2BC)
[013040B0]demokext+000070 (00000001, 2FF3B338)
[001E3BF4]config_kmod+0000F0 (??, ??, ??)
[001E3FA8]sysconfig+000140 (??, ??, ??)
[000039D8].sys_call+000000 ()
[10000570]main+000280 (??, ??)
[10000188]__start+000088 ()
```

3. To step forward four instructions, type `s 4` and press Enter.
4. Reexamine the stack by typing `stack` and pressing Enter.

   It should now include the **strlen** call and should look similar to the following:

```
thread+001800 STACK:
[01304500]strlen+000000 ()
[013042CC]write_log+000028 (10002040, 2FF3B258, 2FF3B2BC)
[013040B0]demokext+000070 (00000001, 2FF3B338)
[001E3BF4]config_kmod+0000F0 (??, ??, ??)
[001E3FA8]sysconfig+000140 (??, ??, ??)
[000039D8].sys_call+000000 ()
[10000570]main+000280 (??, ??)
[10000188]__start+000088 ()
```

5. If you do not see the **strlen** function call, continue stepping until it is displayed.
6. Toggle the KDB kernel debugger option to display the top 64 bytes for each stack frame by typing `set display_stack_frames` and pressing Enter.
7. Display the stack again with the **display_stack_frames** option turned on by typing `stack` and pressing Enter.

   The output should be similar to the following:

```
thread+001800 STACK:
[01304510]strlen+000000 ()
=================================================================
2FF3B1C0: 2FF3 B210  2FF3 B380  0130 4364  0000 0000   /.../....0Cd....
2FF3B1D0: 2FF3 B230  0130 4754  0023 AD5C  2222 2082   /..0.0GT.#.\"" .
2FF3B1E0: 0012 0000  2FF3 B400  0000 0480  0000 510C   ..../.........Q.
2FF3B1F0: 2FF3 B260  4A22 2860  001D CEC8  0000 153C   /..`J"(`.......<
=================================================================
[013042CC]write_log+000028 (10002040, 2FF3B258, 2FF3B2BC)
=================================================================
2FF3B210: 2FF3 B2E0  0000 0003  0130 40B4  0000 0000   /........0@.....
2FF3B220: 0000 0000  2FF3 B380  1000 2040  2FF3 B258   ..../..... @/..X
```

```
2FF3B230: 2FF3 B2BC  0000 0000  001E 5968  0000 0000   /.........Yh....
2FF3B240: 0000 0000  0027 83E8  0048 5358  007F FFFF   .....'...HSX....
=======================================================================
[013040B0]demokext+000070 (00000001, 2FF3B338)
=======================================================================
2FF3B2E0: 2FF3 B370  2233 4484  001E 3BF8  0000 0000   /..p"3D...;.....
2FF3B2F0: 0000 0000  0027 83E8  0000 0001  2FF3 B338   .....'....../..8
2FF3B300: E300 1E30  0000 0020  2FF1 F9F8  2FF1 F9FC   ...0... /.../...
2FF3B310: 8000 0000  0000 0001  2FF1 F780  0000 3D20   ......../.....=
[001E3BF4]config_kmod+0000F0 (??, ??, ??)
=======================================================================
2FF3B370: 2FF3 B3C0  0027 83E8  001E 3FAC  2FF2 2FF8   /....'....?././.
2FF3B380: 0000 0002  2FF3 B400  F014 8912  0000 0FFE   ..../...........
2FF3B390: 2FF3 B388  0000 153C  0000 0001  2000 7758   /......<.... .wX
2FF3B3A0: 0000 0000  0000 09B4  0000 0FFE  0000 0000   ................
=======================================================================
[001E3FA8]sysconfig+000140 (??, ??, ??)
=======================================================================
2FF3B3C0: 2FF2 1AA0  0002 D0B0  0000 39DC  2222 2022   /.........9."" "
2FF3B3D0: 0000 3E7C  0000 0000  2000 9CF8  2000 9D08   ..>|.... ... ...
2FF3B3E0: 2000 A1D8  0000 0000  0000 0000  0000 0000    ...............
2FF3B3F0: 0000 0000  0024 FA90  0000 0000  0000 0000   .....$..........
=======================================================================
[000039D8].sys_call+000000 ()
=======================================================================
2FF21AA0: 2FF2 2D30  0000 0000  1000 0574  0000 0000   /.-0.......t....
2FF21AB0: 0000 0000  2000 0B14  2000 08AC  2FF2 1AE0   .... ... .../...
2FF21AC0: 0000 000E  F014 992D  6F69 6365  3A20 0000   .......-oice: ..
2FF21AD0: FFFF FFFF  D012 D1C0  0000 0000  0000 0000   ................
=======================================================================
[10000570]main+000280 (??, ??)
=======================================================================
2FF22D30: 0000 0000  0000 0000  1000 018C  0000 0000   ................
2FF22D40: 0000 0000  0000 0000  0000 0000  0000 0000   ................
2FF22D50: 0000 0000  0000 0000  0000 0000  0000 0000   ................
2FF22D60: 0000 0000  0000 0000  0000 0000  0000 0000   ................
=======================================================================
[10000188]__start+000088 ()
```

The displayed data can be interpreted using the diagram displayed in the Subroutine Linkage Conventions section of the *Assembler Language Reference* book.

8. Toggle the **display_stack_frames** option off by typing `set display_stack_frames` and pressing Enter.
9. Toggle the KDB kernel debugger option to display the registers saved in each stack frame by typing `set display_stacked_regs` and pressing Enter.
10. Display the stack again with the **display_stacked_regs** option activated by typing `stack` and pressing Enter.

    The display should be similar to the following:

```
thread+001800 STACK:
[01304510]strlen+000010 ()
[013042CC]write_log+000028 (10002040, 2FF3B258, 2FF3B2BC)
   r30 : 00000000 r31 : 01304648
[013040B0]demokext+000070 (00000001, 2FF3B338)
   r30 : 00000000 r31 : 00000000
[001E3BF4]config_kmod+0000F0 (??, ??, ??)
   r30 : 00000005 r31 : 2FF21AF8
[001E3FA8]sysconfig+000140 (??, ??, ??)
   r30 : 04DAE000 r31 : 00000000
[000039D8].sys_call+000000 ()
[10000570]main+000280 (??, ??)
   r25 : DEADBEEF r26 : DEADBEEF r27 : DEADBEEF r28 : DEADBEEF r29 : DEADBEEF
   r30 : DEADBEEF r31 : DEADBEEF
[10000188]__start+000088 ()
```

11. Toggle the **display_stacked_regs** option off by typing `set display_stacked_regs` and pressing Enter.

## Verbose stack output

To see more information about stack outputs, do the following:

1. Display the stack in raw format by typing `dw @r1 90` and pressing Enter:

   **Note:** The address for the stack is in general purpose register 1. The address can be obtained from the output when the **display_stack_frames** option is set.

   This subcommand displays `0x90` words of the stack in hexadecimal and ASCII. The output should be similar to the following:

```
2FF3B1C0: 2FF3B210 2FF3B380 01304364 00000000  /.../....0Cd....
2FF3B1D0: 2FF3B230 01304754 0023AD5C 22222082  /..0.0GT.#.\"" .
2FF3B1E0: 00120000 2FF3B400 00000480 0000510C  ..../.........Q.
2FF3B1F0: 2FF3B260 4A222860 001DCEC8 0000153C  /..`J"(`.......<
2FF3B200: 00000000 00000000 00000000 01304648  .............0FH
2FF3B210: 2FF3B2E0 00000003 013040B4 00000000  /........0@.....
2FF3B220: 00000000 2FF3B380 10002040 2FF3B258  ..../..... @/..X
2FF3B230: 2FF3B2BC 00000000 001E5968 00000000  /.........Yh....
2FF3B240: 00000000 002783E8 00485358 007FFFFF  .....'...HSX....
2FF3B250: 10002040 00000000 64656D6F 6B657874  .. @....demokext
2FF3B260: 20776173 2063616C 6C656420 666F7220   was called for
2FF3B270: 636F6E66 69677572 6174696F 6E0A0000  configuration...
2FF3B280: 00000000 00000000 00001000 2FF3B390  ............/...
2FF3B290: 2FF3B2E0 00040003 001CE9EC 314C0000  /...........1L..
2FF3B2A0: 2FF3B2E0 002783E8 2FF3B338 00000000  /....'../..8....
2FF3B2B0: 00000000 2E746578 74000000 10000100  .....text.......
2FF3B2C0: 10000100 00000710 00000100 00000000  ................
2FF3B2D0: 00000000 2FF3B380 00000000 00000000  ..../...........
2FF3B2E0: 2FF3B370 22334484 001E3BF8 00000000  /..p"3D...;.....
2FF3B2F0: 00000000 002783E8 00000001 2FF3B338  .....'....../..8
2FF3B300: E3001E30 00000020 2FF1F9F8 2FF1F9FC  ...0... /.../...
2FF3B310: 80000000 00000001 2FF1F780 00003D20  ......../.....=
2FF3B320: 2FF21AE8 00000010 01304748 00000001  /........0GH....
2FF3B330: 2FF21AE8 00000010 2FF3B320 FFFFFFFF  /......./.. ....
2FF3B340: 00000001 00000000 00000000 00000000  ................
2FF3B350: 00000010 00001C08 00000000 00000000  ................
2FF3B360: 00000031 82222824 00000005 2FF21AF8  ...1."($..../...
2FF3B370: 2FF3B3C0 002783E8 001E3FAC 2FF22FF8  /....'....?././.
2FF3B380: 00000002 2FF3B400 F0148912 00000FFE  ..../...........
2FF3B390: 2FF3B388 0000153C 00000001 20007758  /......<.... .wX
2FF3B3A0: 00000000 000009B4 00000FFE 00000000  ................
2FF3B3B0: 00000010 E6001800 04DAE000 00000000  ................
2FF3B3C0: 2FF21AA0 0002D0B0 000039DC 22222022  /.........9."" "
2FF3B3D0: 00003E7C 00000000 20009CF8 20009D08  ..>|.... ... ...
2FF3B3E0: 2000A1D8 00000000 00000000 00000000   ..............
2FF3B3F0: 00000000 0024FA90 00000000 00000000  .....$.........
```

   The displayed data can be interpreted using the diagram displayed in the Subroutine Linkage Conventions section of the *Assembler Language Reference* book.

2. Clear all breakpoints by typing the following:

   ```
   ca
   ```

3. Exit the kernel debugger by typing `g` on the command line. Upon exiting the debugger, the prompt from the demo program is displayed. The default prompt is `./demo`.
4. Enter a choice of `0` to unload the kernel extension and quit the KDB kernel debugger.

# Chapter 5. Setting breakpoints

The KDB kernel debugger creates a table of breakpoints that it maintains. When a breakpoint is set, the debugger temporarily replaces the corresponding instruction with the trap instruction. The instruction overlaid by the breakpoint operates when you issue any subcommand that would cause that instruction to be initiated.

For more information on setting or clearing breakpoints, see Chapter 17, "Breakpoint and steps subcommands," on page 115.

Setting a breakpoint is essential for debugging kernel extensions. The general steps for setting a breakpoint are the following:

1. Locate the assembler instruction corresponding to the C statement of the kernel system that you are debugging.

   The process of locating the assembler instruction and obtaining its offset is explained in Chapter 3, "Viewing and modifying global data," on page 13.
2. Get the offset of the assembler instruction from the listing.
3. Locate the address where the kernel extension is loaded.
4. Add the address of the assembler instruction to the address where kernel extension is loaded.
5. Set the breakpoint with the KDB **b** (break) subcommand.

**Note:** To continue with the **demokext** example, set a break at the C source line 67, which increments the *demokext_j* variable. The list file indicates that this line starts at an offset of 0xE0.

The specific steps for setting a breakpoint are included in the following methods:

- "Method 1: Using the lke subcommand"
- "Method 2: Using the nm subcommand" on page 22
- "Method 3: Using the kmid pointer" on page 23
- "Method 4: Using the devsw subcommand" on page 23

## Method 1: Using the lke subcommand

The KDB **lke** subcommand displays a list of loaded kernel extensions. To find the address of the modules for a particular extension use the KDB subcommand **lke** *entry_number*, where *entry_number* is the extension number of interest. A list of Process Trace Backs that shows the beginning addresses of routines contained in the extension is in the displayed data.

**Note:** The default prompt is `KDB(0)>`.

1. Determine the address where the kernel extension is loaded. For information about how to do this, see Chapter 3, "Viewing and modifying global data," on page 13.
2. List all loaded extensions by typing **lke** on the command line.

   The results should be similar to the following:

```
   ADDRESS     FILE FILESIZE    FLAGS MODULE NAME

1 04E17F80 01303F00 000007F0 00000272  ./demokext
2 04E17E80 0503A000 00000E88 00000248  /unix
3 04E17C00 04FA3000 00071B34 00000272  /usr/lib/drivers/nfs.ext
4 04E17A80 05021000 00000E88 00000248  /unix
5 04E17800 01303B98 00000348 00000272  /usr/lib/drivers/nfs_kdes.ext
6 04E17B80 04F96000 00000E34 00000248  /unix
7 04E17500 01301A10 0000217C 00000272  /etc/drivers/blockset64
                     .
                     .
```

 

Enter the Ctrl+C key sequence to exit the KDB kernel debugger paging function. Pressing Enter displays the next page of data. Pressing the Spacebar displays the next line of data. The number of lines per page can be changed by typing `set screen_size nn` on the command line where `nn` is the number of lines per page.

3. List detailed information about the extension of interest.

   The parameter to the **lke** subcommand is the slot number for the *./demokext* entry from the previous step. To display information for slot 1, type the following on the command line:

   ```
   lke 1
   ```

   The output from this command is similar to:

   ```
        ADDRESS     FILE FILESIZE    FLAGS MODULE NAME

     1 04E17F80 01303F00 000007F0 00000272  ./demokext
   le_flags....... TEXT KERNELEX DATAINTEXT DATA DATAEXISTS
   le_next........ 04E17E80 le_fp.......... 00000000
   le_filename.... 04E17FD8 le_file........ 01303F00
   le_filesize.... 000007F0 le_data........ 013045C8
   le_tid......... 00000000 le_datasize.... 00000128
   le_usecount.... 00000003 le_loadcount... 00000001
   le_ndepend..... 00000001 le_maxdepend... 00000001
   le_ule......... 0502E000 le_deferred.... 00000000
   le_exports..... 0502E000 le_de.......... 6C696263
   le_searchlist.. B0000420 le_dlusecount.. 00000000
   le_dlindex..... 00002F6C le_lex......... 00000000
   le_fh.......... 00000000 le_depend.... @ 04E17FD4
   TOC@........... 013046D4
                               <PROCESS TRACE BACKS>
                      .demokext 01304040                  .close_log 013041FC
                     .write_log 01304240                   .open_log 013042B4
                        .strcpy 01304320              .sprintf.glink 01304428
                .fp_close.glink 01304450                     .strlen 01304480
                .fp_write.glink 01304578              .fp_open.glink 013045A0
   ```

   From the PROCESS TRACE BACKS, you can see that the first instruction of **demokext** is at `01304040`. The break for line 67 would be at this address plus E0.

4. Set the break at the desired location by typing the following:

   ```
   b 01304040+e0
   ```

   KDB displays the address at which the breakpoint is located.

5. Clear all breakpoints by typing the following:

   ```
   ca
   ```

## Method 2: Using the nm subcommand

If the kernel extension is not stripped, the KDB kernel debugger can be used to locate the address of the load point by name. For example, the **nm demokext** subcommand returns the address of the **demokext** routine after it is loaded. This address can then be used to set a breakpoint.

**Note:** The default prompt is `KDB(0)>`.

1. To translate a symbol to an effective address, type the following:

   ```
   nm demokext
   ```

   The output is similar to the following:

   ```
   Symbol Address : 01304040
      TOC Address : 013046D4
   ```

The value of the **demokext** symbol is the address of the first instruction of the **demokext** routine. This value can be used to set a breakpoint.

2. Set the break at the desired location by typing the following:
```
b 01304040+e0
```
KDB displays the address at which the breakpoint is set.
3. Display the word at the breakpoint by typing the following:
```
dw 01304040+e0
```
The results are similar to the following:
```
01304120: 80830000 30840001 90830000 809F0030  ....0.........0
```
This can be checked against the assembly code in the listing to verify that the break is set to the correct location.
4. Clear all breakpoints by typing the following:
```
ca
```

## Method 3: Using the kmid pointer

To locate the address of the entry point for a kernel extension, use the value of the **kmid** pointer returned by the **sysconfig(SYS_KLOAD)** subroutine when the kernel extension is loaded. The **kmid** pointer points to the address of the load point routine.

To get the address of the load point, print the **kmid** value during the **sysconfig** call from the configuration method. For example, use the **demo.c** module. Then start the KDB kernel debugger and display the value pointed to by the **kmid** pointer.

**Note:** The default prompt is `KDB(0)>`.

1. Display the memory at the address returned as the **kmid** pointer from the **sysconfig** subroutine, by typing the following:
```
dw 1304748
```
KDB kernel debugger responds with something similar to:
```
demokext+000000: 01304040 01304754 00000000 01304648  .0@@.0GT.....0FH
```
The first word of data displayed is the address of the first instruction of the **demokext** routine. The data displayed is at the location `demokext+000000`. This corresponds to line 26 of the map presented earlier. However, `demokext+000000` and `.demokext+000000` are not the same address. The location `.demokext+000000` corresponds to line 10 of the map and is the address of the first instruction for the **demokext** routine.
2. Set the break at the location indicated from the previous command added to the offset to get to line 67 using the following command:
```
b 01304040+e0
```
KDB kernel debugger responds with an indication of the address at which the breakpoint is set.
3. Clear all breakpoints by typing the following:
```
ca
```

## Method 4: Using the devsw subcommand

If the kernel extension is a device driver, use the KDB **devsw** subcommand to locate the desired address. The **devsw** subcommand lists all of the function addresses for the device driver that are in the dev switch table. Usually, the **config** subroutine is the load point routine. For example,

```
MAJ#010  OPEN          CLOSE         READ          WRITE
         0123DE04      0123DC04      0123DB20      0123DA3C
         IOCTL         STRATEGY      TTY           SELECT
         0123D090      01244DF0      00000000      00059774
         CONFIG        PRINT         DUMP          MPX
         0123E8C8      00059774      00059774      00059774
         REVOKE        DSDPTR        SELPTR        OPTS
         00059774      00000000      00000000      00000002
```

**Note:** The default prompt is `KDB(0)>`.

To set a breakpoint, complete the following:

1. Display the device switch table for the first entry by typing the following:

   ```
   devsw 1
   ```

   The KDB kernel debugger **devsw** command displays data similar to the following:

   ```
   Slot address 50006040
   MAJ#001  OPEN          CLOSE         READ          WRITE
            .syopen       .nulldev      .syread       .sywrite
            IOCTL         STRATEGY      TTY           SELECT
            .syioctl      .nodev        00000000      .syselect
            CONFIG        PRINT         DUMP          MPX
            .nodev        .nodev        .nodev        .nodev
            REVOKE        DSDPTR        SELPTR        OPTS
            .nodev        00000000      00000000      00000012
   ```

   **Note:** Because the demonstration program is not a device driver, this example uses the addresses of the first device driver in the device switch table and is not related in any way to the demonstration program.

2. Set a breakpoint at an offset of `0x20` from the beginning of the open routine for the first device driver in the device switch table by typing the following:

   ```
   b .syopen+20
   ```

   KDB kernel debugger displays the location of the break.

3. Clear all breakpoints by typing the following:

   ```
   ca
   ```

4. Turn off symbolic name translation by typing the following:

   ```
   ns
   ```

5. With symbolic name translation turned off, display the device switch table for the first device driver by typing the following:

   ```
   devsw 1
   ```

   The output is similar to the following:

   ```
   Slot address 50006040
   MAJ#001  OPEN          CLOSE         READ          WRITE
            00208858      00059750      002086D4      0020854C
            IOCTL         STRATEGY      TTY           SELECT
            00208290      00059774      00000000      00208224
            CONFIG        PRINT         DUMP          MPX
   ```

6. Set a break at an offset of `0x20` from the beginning of the open routine for the first device driver in the device switch table by typing the following:

   ```
   b 00208858+20
   ```

   This sets the same break that was set at the beginning of this example. KDB displays the location of the break.

7. Toggle symbolic name translation on by typing the following:

```
ns
```

8. Clear all breaks by typing the following:
   ```
   ca
   ```
9. Exit the KDB kernel debugger and let the system resume normal operations by typing the following:
   ```
   g
   ```

# Chapter 6. Using KDB kernel debugger to perform a trace

The **trcpeek** command of KDB kernel debugger allows users to perform a system trace. It allows users to break into the KDB kernel debugger and start, stop, and display a system trace. For more information on system trace, see Trace Facility in *AIX 5L Version 5.3 General Programming Concepts: Writing and Debugging Programs*.

**Note:** The **trcpeek** command is only available through the KDB kernel debugger. It is not available through the **kdb** command.

If the system is in a working state, it is best to use the system trace facility and the **trace** subcommand. The **trcpeek** command is useful when the system is hung and does not respond to terminal input or when the system is initializing and the trace kernel extension is not loaded. The **trcpeek** command can be used to determine where the kernel code is looping. It is also helpful in early system-initialization debugging. For more information, see the **trace** command in *AIX 5L Version 5.3 Commands Reference, Volume 5*.

Only one trace event can be active at a time. A trace can be started from either the system trace facility at the shell prompt, or from KDB Kernel Debugger at the KDB debugger prompt. If a trace is started from the KDB kernel debugger and the system crashes, trace information can be extracted from the dump using the **trcdead** command. For more information, see the **trcdead** command in *AIX 5L Version 5.3 Commands Reference, Volume 5*.

The **trcpeek** command uses the **trcstart**, **trcstop** and **trace** subcommands. For more information, see “trcstart subcommand” on page 380, “trcstop subcommand” on page 381, and “trace subcommand” on page 378.

# Chapter 7. Subcommand lists

You can view an “Alphabetic list” of the subcommands or a “Task category list” on page 38.

The alphabetic list contains columns that identify the following:

- The name of the subcommand and any aliases for the subcommand. The name is linked to complete information about that subcommand.
- A brief description of the subcommand’s function.
- A usage code that identifies when the subcommands can be used.
- Category in which the subcommands are grouped.

The task category list provides the following:

- Links from each task category to the section that lists the subcommands that are used for the task category.
- Links from each of the subcommands in the lists to the complete information for each subcommand. The information includes syntax, description, aliases and examples.

## Alphabetic list

In the following table, the Usage column indicates when each subcommand can be used with the following codes:

| Code | Usage |
|---|---|
| **B** | With *both* the KDB kernel debugger and the **kdb** command |
| **C** | Only with the **kdb** command |
| **K** | Only with the KDB kernel debugger |
| **MP** | An MP kernel (64-bit kernel or 32-bit MP kernel) |
| **64** | Only with 64-bit kernel |
| **32** | Only with 32-bit kernel |

The following table shows the KDB Kernel Debug Program subcommands in alphabetic order:

| Subcommand, aliases | Functions | Usage | Category |
|---|---|---|---|
| ! | Serves as a shell escape and provides a convenient way to run UNIX commands without leaving kdb | K | End user |
| ames | Display VMM address map entries | B | Display VMM information |
| apt | Display VMM APT entries | B | Display VMM information |
| B | Step on branch | K | Breakpoints and steps |
| b, brk | Sets or lists break points | K | Breakpoints and steps |
| bdev, wlm_bdev | Display **wlm** bio devices | B | WLM |
| bmblock, bmblk, bmb | Display Enhanced Journaled File System metadata block | B | Display Enhanced Journaled File System-specific file system information |
| bqueue, wlm_bq | Display **wlm** bio queues | B | WLM |
| bt | Set or list trace points | K | Debugger trace points |

| Subcommand, aliases | Functions | Usage | Category |
| --- | --- | --- | --- |
| btac | Branch target | K | Branch target (IABR) |
| buffer, buf | Display buffer | B | Display general file system and Journal File System information |
| buserr | PCI bus error injection | K | PCI cfg space and I/O debugging |
| businfo | Display structure businfo | B | PCI cfg space and I/O debugging |
| c, cl | Clear break point | K | Breakpoints and steps |
| ca | Clear all break points | K | Breakpoints and steps |
| cat | Clear all trace points | K | Debugger trace points |
| cbtac | Clear branch target | K | Branch target (IABR) |
| cdt | Display cdt | C | System trace, dump, and error log |
| check | Run consistency checkers on kernel data structures | B | System trace, dump, and error log |
| cla, class | Display **wlm** class | B | WLM |
| clk, cpl | Display complex lock | B | Locks |
| conv | Base conversion | B | Leaving |
| cpu | Switch to cpu | B, MP | Changing context |
| cr, crid | Display crid table | B | Display context information |
| cred | Display credentials structure | B | Display context information |
| ct | Clear trace point | K | Debugger trace points |
| ctx, context | Switch to KDB context | B, MP | Changing context |
| cupboard | Display NFS cupboard | B | Display NFS information |
| cw | Clear watch | K | Watch DABR |
| d, dump | Display byte data | B | Memory register display and decode |
| dbat | Display dbats | B | Address translation |
| dbgopt | Enable or disable debug options | K | End user |
| dc, dis | Display code | B | Memory register display and decode |
| dcal | Calculate or convert a decimal expression | B | Calculator / converter |
| dd | Display double word data | B | Memory register display and decode |
| ddpb | Display device byte | K | Memory register display and decode |
| ddpd | Display device double word | K | Memory register display and decode |
| ddph | Display device half word | K | Memory register display and decode |
| ddpw | Display device word | K | Memory register display and decode |
| ddvb, diob | Display device byte | K | Memory register display and decode |

| Subcommand, aliases | Functions | Usage | Category |
|---|---|---|---|
| ddvd, diod | Display device double word | K | Memory register display and decode |
| ddvh, dioh | Display device half word | K | Memory register display and decode |
| ddvw, diow | Display device word | K | Memory register display and decode |
| debug | Enable or disable debug | B | End user |
| devsw, dev | Display devsw table | B | Display miscellaneous kernel data structures |
| devnode, devno | Display devnode | B | Display general file system and Journal File System information |
| di, decode | Decode the given instruction | B | Memory register display and decode |
| dla | Checks the system for deadlocks and displays details on threads waiting on locks | B, 64 | Locks |
| dlk | Display dist lock | B, 64 | Locks |
| dnlc, ncache | Display name cache | B | Display general file system and Journal File System information |
| dp | Display byte data | B | Memory register display and decode |
| dpc | Display code | B | PCI cfg space and I/O debugging |
| dpcib | Display PCI configuration space in bytes | K | PCI cfg space and I/O debugging |
| dpcih | Display PCI configuration space in half words | K | PCI cfg space and I/O debugging |
| dpciw | Display PCI configuration space in words | K | PCI cfg space and I/O debugging |
| dpd | Display double word data | B | Memory register display and decode |
| dpw | Display word data | B | Memory register display and decode |
| dr | Display registers | B | Memory register display and decode |
| drlist | Display DRlist | B | Display VMM information |
| drvars, drv | Display DRvars | B, MP | Display miscellaneous kernel data structures |
| dtree, dt | Display Enhanced Journaled File System dtree | B | Display Enhanced Journaled File System-specific file system information |
| dw | Display word data | B | Memory register display and decode |
| e, q, g | Exit | B | Leaving |
| errpt | Display error log entries | B | System trace, dump, and error log |

| Subcommand, aliases | Functions | Usage | Category |
|---|---|---|---|
| exp | List export tables | B | Loader |
| ext | Extract pattern | B | Memory search and extract |
| extp | Extract pattern | B | Memory search and extract |
| f, stack, where | Stack frame trace | B | Common basic display |
| fbuffer, fb | Display freelist | B | Display general file system and Journal File System information |
| fifono, fifonode | Display fifonode | B | Display general file system and Journal File System information |
| file | Display file | B | Display general file system and Journal File System information |
| find | Find symbolic pattern | B | Memory search and extract |
| findp | Find physical address pattern | B | Memory search and extract |
| frameset, frs | Display frame sets | B | Display VMM information |
| free | Count and display free frames | B | Display VMM information |
| freelist | Display free list | B | Display VMM information |
| gfs | Display gfs | B | Display general file system and Journal File System information |
| gnode, gno | Display gnode | B | Display general file system and Journal File System information |
| gt | Go until address | K | Breakpoints and steps |
| h, ?, help | Help | B | End user |
| halt | Halt the machine | K | Leaving |
| hbuffer, hb | Display buffer hash | B | Display general file system and Journal File System information |
| hcal, cal | Calculate or convert a hexadecimal expression | B | Calculator / converter |
| hdnlc, hncache | Display hash and ncache | B | Display general file system and Journal File System information |
| heap, hp | Display kernel heap | B | Display memory allocator information |
| hinode, hino | Display inodehash | B | Display general file system and Journal File System information |
| his, hi, hist | Print history | B | End user |
| hnode, hno | Display hnodehash | B | Display general file system and Journal File System information |
| hvnc, hvcache | Display hash, vcache | B | Display general file system and Journal File System information |
| ibat | Display ibats | B | Address translation |

| Subcommand, aliases | Functions | Usage | Category |
| --- | --- | --- | --- |
| icache, fino | Display icache list | B | Display general file system and Journal File System information |
| ifnet | Display interface | B | Network |
| inode, ino | Display inode | B | Display general file system and Journal File System information |
| inode2, i2 | Display Enhanced Journaled File System inode | B | Display Enhanced Journaled File System-specific file system information |
| intr | Display int handler | B | Display miscellaneous kernel data structures |
| ipc | Display IPC information | B | Display VMM information |
| ipl | Display IPL process information | B | Display miscellaneous kernel data structures |
| j2, jfs2 | Display Enhanced Journaled File System buffer data | B | Display Enhanced Journaled File System-specific file system information |
| j2logbuf | Display JFS2 log buffer structure | B | Display Enhanced Journaled File System-specific file system information |
| j2logx | Display Enhanced Journaled File System logx structure | B | Display Enhanced Journaled File System-specific file system information |
| j2log | Display Enhanced Journaled File System log structure | B | Display Enhanced Journaled File System-specific file system information |
| j2no, jfs2node | Display jfs2node | B | Display Enhanced Journaled File System-specific file system information |
| j2trace, j2trc, j2t | Display Enhanced Journaled File System trace table | B | Display Enhanced Journaled File System-specific file system information |
| kfset, kfs | Display the **kdm** fset cache data structure | B | Display general file system and Journal File System information |
| kmbucket, bucket | Display kmembuckets | B | Display memory allocator information |
| kmstats | Display kmemstats | B | Display memory allocator information |
| ksp | Display KSP region information | B | Display VMM information |
| kvn, kvnode | Display kdm vnode | B | Display general file system and Journal File System information |

| Subcommand, aliases | Functions | Usage | Category |
|---|---|---|---|
| lastbackt | Display lastbackt | B | Display context information |
| lb, lbrk | Sets or lists local breakpoints | K | Breakpoints and steps |
| lbtac | Display local branch target | K | Branch target (IABR) |
| lc, lcl | Clear local breakpoints | K | Breakpoints and steps |
| lcbtac | Clear local branch target | K | Branch target (IABR) |
| lcw | Clear local watch | K | Watch DABR |
| lk | Display lock_t lock | B | Locks |
| lke | List loaded extensions | B | Loader |
| lle | List loader entries | B | Loader |
| lka, lockanch tblk | Display VMM lock anchor or tblock | B | Display VMM information |
| lkh, lockhash | Display VMM lock hash | B | Display VMM information |
| lkw, lockword | Display VMM lock word | B | Display VMM information |
| lq, lockq | Display lock queues | B | Display context information |
| lrustate, lru | Display the lru daemon control variables | B | Display VMM information |
| lvol | Display logical volume | B | Display storage subsystem information |
| lwr | Local stop on read data | K | Watch DABR |
| lwrw | Local stop on read/write data | K | Watch DABR |
| lww | Local stop on write data | K | Watch DABR |
| m | Modify sequential bytes | K | Memory modification |
| mbuf | Display mbuf | B | Network |
| md | Modify sequential double word | K | Memory modification |
| mdbat | Modify dbats | B | Address translation |
| mdpb | Modify device byte | K | Memory modification |
| mdpd | Modify device double word | K | Memory modification |
| mdph | Modify device half | K | Memory modification |
| mdpw | Modify device word | K | Memory modification |
| mdvb, miob | Modify device byte | K | Memory modification |
| mdvd, miod | Modify device double word | K | Memory modification |
| mdvh, mioh | Modify device half | K | Memory modification |
| mdvw, miow | Modify device word | K | Memory modification |
| meml, memlock | Displays information about the memory lock entries | B | Display context information |
| mempool, memp | Display memory pools | B | Display VMM information |
| mibat | Modify ibats | B | Address translation |
| mp | Modify sequential bytes | K | Memory modification |
| mpcib | Modify PCI configuration space in bytes | K | PCI cfg space and I/O debugging |
| mpcih | Modify PCI configuration space in half words | K | PCI cfg space and I/O debugging |
| mpciw | Modify PCI configuration space in words | K | PCI cfg space and I/O debugging |

| Subcommand, aliases | Functions | Usage | Category |
|---|---|---|---|
| mpd | Modify sequential double word | K | Memory modification |
| mpw | Modify sequential word | K | Memory modification |
| mr | Modify registers | B | Memory modification |
| mslb | Modify SLB entry | B | Address translation |
| mst | Display MST area | B | Display context information |
| mtrace | Display information on the Lightweight Memory Trace (LMT) | B | System trace, dump, and error log |
| mw | Modify sequential word | K | Memory modification |
| n, nexti | Next instruction | K | Breakpoints and steps |
| ndd | Display network and device driver statistics | B | Network |
| netm | Display the **net_malloc** event records | B | Network |
| netstat | Display network status | C | Network |
| nsdbg | Display ns_alloc and free event records stored in the kernel. | C | Network |
| nm | Translate symbol to an effective address | B | Namelists and symbols |
| ns | No symbol mode (toggle) | B | Namelists and symbols |
| pbuf | Display physical buf | B | Display storage subsystem information |
| pdt | Display VMM paging device table | B | Display VMM information |
| pfhdata | Display VMM control variables | B | Display VMM information |
| pft | Display VMM PFT entries | B | Display VMM information |
| pgbuf | Display Enhanced Journaled File System pager buffer | B | Display Enhanced Journaled File System-specific file system information |
| pgobj | Display Enhanced Journaled File System pagerObject | B | Display Enhanced Journaled File System-specific file system information |
| pile | Display pile | B | Display Enhanced Journaled File System-specific file system information |
| pnda | Display PNDA area | B | Display context information |
| ppda | Display per processor data area | B | Display context information |
| pr, print | Print a formatted structure at an address | B | Common basic display |
| pta | VMM PTA segment | B | Display VMM information |
| pte | VMM PTE entries | B | Display VMM information |
| pvlist, pvt | Display VMM PVT and PVLIST entries | B | Display VMM information |
| pvol | Display physical volume | B | Display storage subsystem information |
| r, return | Go to end of function | K | Breakpoints and steps |
| reboot | Reboot the machine | K | Leaving |
| rmap | Display VMM RMAP | B | Display VMM information |
| rmst | Remove symbol table | B | Loader |

| Subcommand, aliases | Functions | Usage | Category |
| --- | --- | --- | --- |
| route | Display route | B | Network |
| rq, runq | Display run queues | B | Display context information |
| rqi, rqa | Display RQ information | B | Display context information |
| rtentry | Display rtentry structure | B | Network |
| rtipc | Display RT IPC information | B | Display VMM information |
| rtipcd | Display RT IPCD information | B | Display VMM information |
| rules, rule | Display **wlm** rules | B | WLM |
| runcpu | Allows any other **kdb** subcommand to be automatically run | B | Changing context |
| rvsid | Display reserved vsid information | B, 64 | Display VMM information |
| rxnode | Display radix_node structure | B | Network |
| s, stepi | Single step | K | Breakpoints and steps |
| S | Step on block or blockr | K | Breakpoints and steps |
| scb | Display VMM segment control blocks | B | Display VMM information |
| scd, scdisk | Display scdisk | B | Display storage subsystem information |
| segst64 | Display VMM SEGSTATE | B | Display VMM information |
| set, setup | Display or update kdb toggles | B | End user |
| slab | Display slab | B | Display Enhanced Journaled File System-specific file system information |
| slb | Display SLB entry | B | Address translation |
| slk, spl | Display simple lock | B | Locks |
| sock | Display socket | B | Network |
| sockcup | Display NFS sockcup | B | Display NFS information |
| sockinfo, si | Display socket info by address | B | Network |
| sockpint | Display NFS sockcup | B | Display NFS information |
| specnode, specno | Display specnode | B | Display general file system and Journal File System information |
| sr64 | Display VMM segment region | B, MP | Display VMM information |
| start | Start cpu | K, MP | CPU start and stop |
| stat | System status messages | B | Common basic display |
| status | Processor status | B | Common basic display |
| st | Store one address word in memory | K | Memory modification |
| stbl | List loaded symbol tables | B | Address translation |
| stc | Store one address byte in memory | K | Memory modification |
| ste | Display VMM STAB | B | Display VMM information |
| sth | Store one half-word in memory address half-word | K | Memory modification |
| stop | Stop cpu | KMP | CPU start and stop |
| svcxprt | Display NFS SVCXPRT | B | Display NFS information |

| Subcommand, aliases | Functions | Usage | Category |
| --- | --- | --- | --- |
| svmon | Display information about the memory and paging space usage on a per process basis | C | Display context information |
| swhat | Display VMM SWHAT entries | B | Display VMM information |
| sw, switch | Switch to thread | B | Changing context |
| symptom | Display symptom string for a dump | C | Common basic display |
| tcb | Display TCBs | B | Network |
| tcpcb | Display TCP CB | B | Network |
| test, [ | Displays bt condition | K | Debugger trace points |
| time | Display elapsed time | K | Time |
| tpid, th_pid | Display displays all thread entries for a process | B | Display context information |
| tr | Translate to real address | B | Address translation |
| trace | Display trace buffer | B | System trace, dump, and error log |
| trb | Display system timer request blocks | B | Time |
| trcstart | Starts the system trace | K | System trace, dump, and error log |
| trcstop | Stops the system trace | K | System trace, dump, and error log |
| tree | Display Enhanced Journaled File System tree | B | Display Enhanced Journaled File System-specific file system information |
| ts | Translate eaddr to symbol | B | Namelists and symbols |
| ttid, th_tid | Display displays the thread table entry for a specific thread | B | Display context information |
| tv | Display MMU translation | B | Address translation |
| txblock, txblk | Display Enhanced Journaled File System txBlock | B | Display Enhanced Journaled File System-specific file system information |
| txblocki, txblki | Display Enhanced Journaled File System index of txBlock | B | Display Enhanced Journaled File System-specific file system information |
| txlock, txlck | Display Enhanced Journaled File System txLock | B | Display Enhanced Journaled File System-specific file system information |
| udb | Display UDBs | B | Network |
| var | Display var | B | Display miscellaneous kernel data structures |
| varlist | List user variables | B | End user |
| varrm, unalias | Remove user variable | B | End user |
| varset, alias | Define a user variable | B | End user |
| vfs, mount | Display vfs | B | Display general file system and Journal File System information |

| Subcommand, aliases | Functions | Usage | Category |
|---|---|---|---|
| vmdmap | VMM disk map | B | Display VMM information |
| vmlocks | VMM spin locks | B | Display VMM information |
| vmaddr | VMM Addresses | B | Display VMM information |
| vmbufst | Display dump buffer structures | B | Display VMM information |
| vmint | Display VMM vmintervls information | B | Display VMM information |
| vmker | Display VMM kernel segment data | B | Display VMM information |
| vmlocks, vmlock, vl | Display VMM spin locks | B | Display VMM information |
| vmlog | Display VMM error log | B | Display VMM information |
| vmpool | Display VMM resource pools | B | Display VMM information |
| vmstat | Display VMM statistics | B | Display VMM information |
| vmthrpgio | Display THRPGIO commands | B | Display VMM information |
| vmwait | Display VMM wait status | B | Display VMM information |
| vnc, vcache | Display vnode cache | B | Display general file system and Journal File System information |
| vnode, vno | Display vnode | B | Display general file system and Journal File System information |
| volgrp | Display volume group | B | Display storage subsystem information |
| vrld | Display VMM reload xlate table | B | Display VMM information |
| vsidd, sidd | Display VSID dump | B | Display VMM information |
| vsidm, sidm | Display VSID alter | B | Display VMM information |
| which | Display name of kernel source file | C | Namelists and symbols |
| wr | Stop on read data | K | Watch DABR |
| wrw | Stop on r/w data | K | Watch DABR |
| ww | Stop on write data | K | Watch DABR |
| xtree, xt | Display Enhanced Journaled File System xtree | B | Display Enhanced Journaled File System-specific file system information |
| xmalloc, xm | Display heap debug | B | Display memory allocator information |
| zproc | Display VMM zeroing kproc | B | Display VMM information |

## Task category list

The categories in which the subcommands are grouped are as follows:


- Chapter 25, “Address translation subcommands,” on page 241
- Chapter 20, “Branch target subcommands,” on page 137
- Chapter 17, “Breakpoint and steps subcommands,” on page 115
- Chapter 10, “Changing context subcommands,” on page 59
- Chapter 11, “Calculator and converter subcommands,” on page 69
- Chapter 13, “Basic display subcommands,” on page 75

- Chapter 12, “CPU start and stop subcommands,” on page 73
- Chapter 27, “Display context information subcommands,” on page 261
- Chapter 30, “Display general and Journal File System (JFS) information subcommands,” on page 305
- Chapter 31, “Display Enhanced Journaled File System information subcommands,” on page 343
- Chapter 29, “Display memory allocation information subcommands,” on page 293
- Chapter 23, “Display kernel data structures subcommands,” on page 155
- Chapter 32, “Display NFS information subcommands,” on page 365
- Chapter 28, “Display storage subsystem information subcommands,” on page 283
- Chapter 24, “Display VMM subcommands,” on page 165
- Chapter 8, “End user subcommands,” on page 41
- Chapter 18, “Debugger trace points subcommands,” on page 127
- Chapter 9, “Leaving kdb subcommands,” on page 55
- Chapter 26, “Loader subcommands,” on page 251
- Chapter 35, “Lock subcommands,” on page 389
- Chapter 16, “Memory modification subcommands,” on page 107
- Chapter 14, “Memory register display and decode subcommands,” on page 89
- Chapter 15, “Memory search and extract subcommands,” on page 101
- Chapter 21, “Namelist and symbols subcommands,” on page 141
- Chapter 36, “Network subcommands,” on page 393
- Chapter 22, “PCI configuration space and I/O debugging subcommands,” on page 145
- Chapter 34, “System trace, dump and error log subcommands,” on page 377
- Chapter 33, “Time subcommands,” on page 371
- Chapter 19, “Watch DABR subcommands,” on page 133
- Chapter 37, “Workload Manager (WLM) subcommands,” on page 421

# Chapter 8. End user subcommands

The subcommands in this category explain how category help works, list and set **kdb** command toggles, and create, display and remove user-defined variables. These subcommands include the following:

- h
- set
- dbgopt
- varset
- varlist
- varrm
- his
- debug
- !

## h subcommand

### Purpose

The **h** subcommand displays a list of all available subcommands in the debugger. When run with a parameter, this list is restricted to only a particular category of subcommands. The list of categories is:


- Chapter 25, “Address translation subcommands,” on page 241
- Chapter 20, “Branch target subcommands,” on page 137
- Chapter 17, “Breakpoint and steps subcommands,” on page 115
- Chapter 10, “Changing context subcommands,” on page 59
- Chapter 11, “Calculator and converter subcommands,” on page 69
- Chapter 13, “Basic display subcommands,” on page 75
- Chapter 12, “CPU start and stop subcommands,” on page 73
- Chapter 27, “Display context information subcommands,” on page 261
- Chapter 30, “Display general and Journal File System (JFS) information subcommands,” on page 305
- Chapter 31, “Display Enhanced Journaled File System information subcommands,” on page 343
- Chapter 29, “Display memory allocation information subcommands,” on page 293
- Chapter 23, “Display kernel data structures subcommands,” on page 155
- Chapter 32, “Display NFS information subcommands,” on page 365
- Chapter 28, “Display storage subsystem information subcommands,” on page 283
- Chapter 24, “Display VMM subcommands,” on page 165
- Chapter 8, “End user subcommands,” on page 41
- Chapter 18, “Debugger trace points subcommands,” on page 127
- Chapter 9, “Leaving kdb subcommands,” on page 55
- Chapter 26, “Loader subcommands,” on page 251
- Chapter 35, “Lock subcommands,” on page 389
- Chapter 16, “Memory modification subcommands,” on page 107
- Chapter 14, “Memory register display and decode subcommands,” on page 89
- Chapter 15, “Memory search and extract subcommands,” on page 101
- Chapter 21, “Namelist and symbols subcommands,” on page 141
- Chapter 36, “Network subcommands,” on page 393
- Chapter 22, “PCI configuration space and I/O debugging subcommands,” on page 145
- Chapter 34, “System trace, dump and error log subcommands,” on page 377
- Chapter 33, “Time subcommands,” on page 371
- Chapter 19, “Watch DABR subcommands,” on page 133
- Chapter 37, “Workload Manager (WLM) subcommands,” on page 421


### Syntax

**h** [*topic*]

### Parameters

- *topic* – specifies the name, or partial name, of a particular help category. If more than one category name matches the topic, only the first matching category and its subcommands are displayed.

### Aliases

**?**, **help**

## Example

The following is an example of how to use the **help** alias for the **h** subcommand:

```
KDB(0)> help user
CMD      ALIAS    ALIAS    FUNCTION                   ARG


                *** end-user ***

h        ?        help     help                       [topic]
set               setup    display/update kdb toggles [toggle]
dbgopt                     enable/disable debug options
varset   alias             define a user variable     var value
varlist                    list user variables
varrm    unalias           remove user variable       var
his      hi       hist     print history              [?][count]
debug                      enable/disable debug       [?]
KDB(0)>
```

## set subcommand

### Purpose

The **set** subcommand lists and sets kdb toggles.

### Syntax

**set** [*toggle* [*value*]]

### Parameters

- *toggle* – Identifies the option to be toggled or set by decimal number or name.
- *value* – Indicates the decimal number or expression to be set for an option.

  **Note:** Some toggles allow the value to be omitted. In that case, the set subcommand cycles the toggle through all of its possible settings.

The values that are valid for the KDB Kernel Debugger and the **kdb** command are the following:

- *no_symbol* – Suppresses symbol name lookup when addresses are displayed.
- *mst_wanted* – Displays all **mst** items in the stack trace subcommand each time an interrupt is detected in the stack. For a shorter display, disable this toggle.
- *screen_size* – Changes the integrated more prompt window size.
- *power_pc_syntax* – Displays PowerPC platform-based instruction mnemonics when enabled (See the 92 and 94subcommands). Displays the old POWER™ family mnemonics when disabled.
- *origin* – Sets the origin variable to the value of the specified expression. Origins are used to match addresses with assembly language listings. Assembly language listings express addresses as offsets from the start of the file.
- *unix_symbols_start_from* – Indicates the lowest effective address from which symbol search is started. To force other values to be displayed in hexadecimal, set this toggle.
- *hexadecimal_wanted* – Applies to **thread** and **process** subcommand. It is possible to have information in decimal form.
- *screen_previous* – Applies to the memory display subcommands, such as **d** and **dw**. To repeat the last memory display subcommand, press Enter at an empty kdb prompt. If *screen_previous* is set to false, memory is displayed at the next higher address. If *screen_previous* is set to true, memory is displayed at the next lower address.
- *display_stack_frames* – Applies to **f** subcommand. When it is true, the **f** subcommand prints a part of the stack in binary mode.
- *display_stacked_regs* – Applies to **f** subcommand. When it is true, the **f** subcommand prints register values saves in the stack.
- *64_bit* – Prints 64-bit registers on 64-bit architecture. By default, only 32-bit formats are printed.
- *ldr_segs_wanted* – Toggles interpretation of effective addresses in segment 11 (0xbxxxxxxx) and segment 13 (0xdxxxxxxx) off and on as references to loader data.
- *trace_back_lookup* – Processes trace back information on user code (text or shared-lib) and kernext code. It can be used to see function names. By default, it is not set.
- *scroll* – Enables or disables the integrated more prompt.
- *edit* – Provides command line editing features similar to those provided by the Korn shell. The mode specified provides editing features similar to editors, such as vi, emacs, and gmacs.

  For example, to turn on vi-style command line editing, type the following at the kdb prompt:

  ```
  set edit vi
  ```
- *default_xmalloc_heap* – Specifies the default heap for the **xmalloc** subcommand. If this option is `0`, the **xmalloc** subcommand uses the kernel heap.

The values that apply only to the **kdb** command are the following:

- *logfile* – Enables or disables logging for a specified log file name. If *logfile* is invoked without a parameter specifying a file name, logging is disabled.
- *loglevel* – Allows you to choose the granularity level of logging. Valid choices are the following:

  **0** off

  **1** Log commands only

  **2** Log commands and output. This is the default.

The options that apply only to the KDB kernel debugger are the following:

- *emacs_window* – Toggles suppression of extra line feeds for running under emacs.
- *local_breakpoint_attach* – Toggles to choose whether local breakpoints are thread or CPU based. By default, on POWER RS1, local breakpoints are CPU-based, and on the POWER-based platform they are thread-based.

  **Note:** This toggle must be accessed using the option number. It cannot be toggled by name.
- *kdb_stop_all_cpu* – Toggles to select whether all processors or a single processor stops when the KDB kernel debugger is invoked.
- *tweq_r1_r1* – Causes the KDB kernel debugger to break on the `tweq r1, r1` instruction. This is the trap instruction reserved for entering LLDB.
- *kext_IF_active* – Toggles to disable and enable subcommands added to the KDB kernel debugger through kernel extensions. By default, all subcommands registered by kernel extensions are active.
- *IPI_enable* – Toggles to control how the KDB kernel debugger notifies other processors to stop when the *KDB stops all processors* value is true. If the *IPI_enable* value is true, the KDB kernel debugger uses inter-processor interrupts. If *IPI_enable* is false, the decrementer interrupt is used.
- *no_brkpt_warning* – Controls whether the KDB kernel debugger prints warning messages when it ignores certain breakpoints, for example, a context mismatch. If the *no_brkpt_warning* value is set to true, the KDB kernel debugger does not print warning messages when it ignores certain breakpoints. If the *no_brkpt_warning* value is set to false, the KDB kernel debugger prints warning messages when it ignores certain breakpoints.

## Aliases

**setup**

## Example

The following is an example of how to use the **set** subcommand:

```
KDB(0)> set
 No toggle name              current value

  1 no_symbol                false
  2 mst_wanted               true
  3 screen_size              24
  4 power_pc_syntax          true
  5 origin                   00000000
  6 unix_symbols_start_from  00001000
  7 hexadecimal_wanted       true
  8 screen_previous          false
  9 display_stack_frames     false
 10 display_stacked_regs     false
 11 64_bit                   true
 12 ldr_segs_wanted          false
 13 emacs_window             false
 14 local_breakpoint_attach  thread
 15 kdb_stop_all_cpu         true
 17 kext_IF_active           true
```

```
 18 trace_back_lookup          false
 19 IPI_enable                 true
 20 scroll                     false
 21 edit                       noedit
 24 no_brkpt_warning           false
 25 default_xmalloc_heap       00000000
KDB(0)> dc waitproc 5
.waitproc+000000     mflr    r0
.waitproc+000004     mfcr    r12
.waitproc+000008      std    r31,FFFFFFF8(stkp)
.waitproc+00000C      std    r30,FFFFFFF0(stkp)
.waitproc+000010      std    r29,FFFFFFE8(stkp)
KDB(0)> set origin 100
  5 origin                     00000100
KDB(0)> dc waitproc 5
.waitproc+000000 (ORG+00026CB8)     mflr    r0
.waitproc+000004 (ORG+00026CBC)     mfcr    r12
.waitproc+000008 (ORG+00026CC0)      std    r31,FFFFFFF8(stkp)
.waitproc+00000C (ORG+00026CC4)      std    r30,FFFFFFF0(stkp)
.waitproc+000010 (ORG+00026CC8)      std    r29,FFFFFFE8(stkp)
KDB(0)> set scroll false
 20 scroll                     false
```

## dbgopt subcommand

### Purpose

The **dbgopt** subcommand toggles low-level tracing options within the kernel.

### Syntax

**dbgopt**

### Parameters

The **dbgopt** subcommand presents a menu that allows the user to enable rc.boot tracing and tracing of exec calls. The tracing enabled by this subcommand is performed using the kernel **printf** function and is unrelated to the system trace facility.

### Aliases

No aliases.

### Example

The following is an example of how to use the **dbgopt** subcommand:

```
KDB(0)> dbgopt
Debug options:
--------------
1. Toggle rc.boot tracing - currently DISABLED
2. Toggle tracing of exec calls - currently DISABLED
q. Exit

Enter option: 2

Debug options:
--------------
1. Toggle rc.boot tracing - currently DISABLED
2. Toggle tracing of exec calls - currently ENABLED
q. Exit

Enter option: q

KDB(0)>
```

## varset subcommand

### Purpose

The **varset** subcommand creates a new user-defined variable.

**Note:** In the KDB kernel debugger, user variables are persistent across invocations of the debugger but not across system reboots. In the **kdb** command, user variables are not persistent across invocations.

### Syntax

**varset** *name* [*value*]

### Parameters

- *name* – Specifies the name of a user variable. If it does not already exist, the variable is created. Otherwise, the value of the existing variable is changed. Variable names are case sensitive and can consist of letters, numbers, and the underscore (_) character.
- *value* – Is a string assigned verbatim to the user variable specified by name. If omitted, the user variable is assigned an empty string. The value can contain spaces.

After a variable is created, any occurrence of the variable name in a subcommand is replaced with the value assigned to that variable.

If any variable substitutions occur, the resulting subcommand is printed between two less than and two greater than signs before it is run. For example, `<<dw kdb_avail 1>>`.

All variable substitutions are done before any additional parsing of the subcommand, and the substitutions are done on a textual basis. This allows a single variable to expand into multiple subcommand parameters.

### Aliases

No aliases.

### Example

The following is an example of how to use the **varset** subcommand:

```
KDB(0)> varset myvar kdb_avail
KDB(0)> dw myvar
<<dw kdb_avail>>
kdb_avail+000000: 00000001 00000000 0800004C 00001C43  ...........L...C
KDB(0)> varset myvar kdb_avail 1
KDB(0)> dw myvar
<<dw kdb_avail 1>>
kdb_avail+000000: 00000001                             ....
KDB(0)>
```

## varlist subcommand

### Purpose

The **varlist** subcommand displays all user-defined variables previously created with the **varset** subcommand.

### Syntax

**varlist**

### Parameters

No parameters.

### Aliases

No aliases.

### Example

The following is an example of how to use the **varlist** subcommand:

```
KDB(0)> varset myvar kdb_avail
KDB(0)> varlist
Slot        Name                             Value
  0        myvar                         kdb_avail
KDB(0)>
```

# varrm subcommand

## Purpose

The **varrm** subcommand removes user-defined variables previously created with the **varset** subcommand.

## Syntax

**varrm** *name*

## Parameters

- *name* – Specifies the user variable to remove. Variable names are case sensitive and consist of letters, numbers, and the underscore (_) character.

## Aliases

No aliases.

## Example

The following is an example of how to use the **varrm** subcommand:

```
KDB(0)> varlist
Slot        Name                             Value
  0        myvar                        kdb_avail
KDB(0)> varrm myvar
KDB(0)> varlist
Slot        Name                             Value
KDB(0)>
```

## his subcommand

### Purpose

The **his** subcommand prints a history of user input. A parameter can be used to specify the number of historical entries to display.

### Syntax

**his** [*value*]

### Parameters

- *value* – Indicates a decimal value or expression indicating the number of previous user entries to display.

Each historical entry can be recalled and edited for use with the usual control characters (as in emacs).

### Aliases

**hi**, **hist**

### Example

No example.

## debug subcommand

### Purpose

The **debug** subcommand prints additional information while the KDB kernel debugger is running to help ensure that the debugger is functioning properly.

### Syntax

**debug** [*options*]

### Parameters

- *options* – Specifies the debug option to be turned on or off. View possible values by specifying the **?** flag.

If the **debug** subcommand is invoked with no parameters, the currently-active debug options are displayed.

### Aliases

No aliases.

### Example

The following is an example of how to use the **debug** subcommand:

```
KDB(4)> debug ? //debug help
vmm HW lookup debug... on with arg 'dbg1++', off with arg 'dbg1--'
vmm tr/tv cmd debug... on with arg 'dbg2++', off with arg 'dbg2--'
vmm SW lookup debug... on with arg 'dbg3++', off with arg 'dbg3--'
symbol lookup debug... on with arg 'dbg4++', off with arg 'dbg4--'
stack trace debug..... on with arg 'dbg5++', off with arg 'dbg5--'
BRKPT debug (list).... on with arg 'dbg61++', off with arg 'dbg61--'
BRKPT debug (instr)... on with arg 'dbg62++', off with arg 'dbg62--'
BRKPT debug (suspend). on with arg 'dbg63++', off with arg 'dbg63--'
BRKPT debug (phantom). on with arg 'dbg64++', off with arg 'dbg64--'
BRKPT debug (context). on with arg 'dbg65++', off with arg 'dbg65--'
DABR debug (address).. on with arg 'dbg71++', off with arg 'dbg71--'
DABR debug (register). on with arg 'dbg72++', off with arg 'dbg72--'
DABR debug (status)... on with arg 'dbg73++', off with arg 'dbg73--'
BRAT debug (address).. on with arg 'dbg81++', off with arg 'dbg81--'
BRAT debug (register). on with arg 'dbg82++', off with arg 'dbg82--'
BRAT debug (status)... on with arg 'dbg83++', off with arg 'dbg83--'
BRKPT debug (context). on  //this debug feature is enabled
KDB(4)> debug dbg5++  //enable debug mode
stack trace debug..... on
KDB(4)> f  //stack frame in debug mode
thread+000180 STACK:
=== Look for traceback at 0x00015278
=== Got traceback at 0x00015280 (delta = 0x00000008)
=== has_tboff = 1, tb_off = 0xD8
=== Trying to find Stack Update Code from 0x000151A8 to 0x00015278
=== Found 0x9421FFA0 at 0x000151B8
=== Trying to find Stack Restore Code from 0x000151A8 to 0x0001527C
=== Trying to find Registers Save Code from 0x000151A8 to 0x00015278
[00015278]waitproc+0000D0 ()
=== Look for traceback at 0x00015274
=== Got traceback at 0x00015280 (delta = 0x0000000C)
=== has_tboff = 1, tb_off = 0xD8
[00015274]waitproc+0000CC ()
=== Look for traceback at 0x0002F400
=== Got traceback at 0x0002F420 (delta = 0x00000020)
=== has_tboff = 1, tb_off = 0x30
[0002F400]procentry+000010 (??, ??, ??, ??)
```

```
/# ls  //Invoke command from command line that calls open
Breakpoint
0024FDE8     stwu   stkp,FFFFFFB0(stkp) stkp=2FF3B3C0,FFFFFFB0(stkp)=2FF3B370
KDB(0)> time  //Report time from leaving the debugger till the break
Command: time  Aliases:
Elapsed time since last leaving the debugger:
2 seconds and 121211136 nanoseconds.
KDB(0)>
```

# ! subcommand

## Purpose

The **!** subcommand serves as a shell escape and provides a way to run UNIX commands without leaving the **kdb** command. This subcommand is only available in the **kdb** command.

**Note:** If output logging is enabled through the *logfile* and *loglevel* **kdb** command options, the output produced by the **!** subcommand is not included in the log file.

## Syntax

**!** [*command*]

## Parameters

- *command* – Passes a command verbatim to a newly spawned UNIX shell for running.

## Aliases

No aliases.

## Example

The following is an example of how to use the **!** subcommand:

```
(0)> ! ls
...            .dtprofile     bin            lib            sbin
.:             .mozilla       dev            lost+found     tftpboot
.TTauthority   .sh_history    dfs            lpp            tmp
.Xauthority    .wmrc          etc            mnt            unix
.bash_history  :              gsa            opt            usr
.dbxhist       TT_DB          home           proc           var
.dt            audit          krb5           project
(0)>
```

# Chapter 9. Leaving kdb subcommands

The subcommands in this category are used to exit the **kdb** command and the KDB kernel debugger, shutdown the machine and reboot the machine. These subcommands include the following:

- e
- reboot
- halt

# e subcommand

## Purpose

The **e** subcommand exits the **kdb** command and KDB kernel debugger.

## Syntax

**e** [*dump*]

## Parameters

- *dump* – Indicates that a system dump will be created when you exit the KDB kernel debugger. The optional *dump* parameter is only applicable to the KDB kernel debugger. The *dump* argument can be specified to force an operating system dump. The method used to force a dump depends on how the KDB kernel debugger was invoked.

  The KDB kernel debugger can be invoked in the following ways:

  **panic** If the KDB kernel debugger was invoked by the **panic** call, force the dump by typing `q dump` and pressing Enter. If another processor enters the KDB kernel debugger after that (for example, a spin-lock timeout), exit the KDB kernel debugger.

  When the dump is complete, control is returned to the KDB kernel debugger and the LEDs show `xxxx`.

  **halt_display**
  If the KDB kernel debugger was invoked by a halt display (C20 on the LED), type `q` and press Enter.

  When the dump is complete, the LEDs show `888 102 700 0c0`.

  **soft_reset**
  If the debugger was invoked by a soft reset (that is, pressing the reset button once), complete the following:

  1. Move the key on the server.

     If the key was in the SERVICE position at boot time, move it to the NORMAL position. Otherwise, move the key to the SERVICE position.

     **Note:** Forcing a dump using this method requires that you know what the key position was at boot time.
  2. Type `quit` and press Enter.

     Do this once for each CPU.

  **break in**
  You cannot create a dump if the debugger was invoked with the break method (^\).

  When the dump is in progress, `_0c9` displays on the LEDs while the dump is copied to disk `hd7` or disk `hd6`.

  The **e** subcommand allows you to exit the KDB kernel debugger session and return to the system with all breakpoints installed in memory. To leave KDB kernel debugger without breakpoints, use the **ca** subcommand.

## Aliases

**q**, **g**

## Example

No example.

# reboot subcommand

## Purpose

The **reboot** subcommand reboots the machine. This subcommand issues a prompt for confirmation that a reboot is desired before beginning the reboot.

**Note:** This subcommand is only available within the KDB kernel debugger. It is not included in the **kdb** command.

## Syntax

**reboot**

## Parameters

No parameters.

## Aliases

No aliases.

## Example

The following is an example of how to use the **reboot** subcommand:

```
KDB(0)> reboot  //reboot the machine
Do you want to continue system reboot? (y/[n]):> y
Rebooting ...
```

# halt subcommand

## Purpose

The **halt** subcommand shuts down the machine.

**Note:** This subcommand is only available within the KDB kernel debugger. It is not included in the **kdb** command.

## Syntax

**halt**

## Parameters

No parameters.

## Aliases

No aliases.

## Example

The following is an example of how to use the **halt** subcommand:

```
KDB(0)> halt
Halting...
```

# Chapter 10. Changing context subcommands

The subcommands in this category are used to change the context that is being debugged. These subcommands include the following:

- sw
- cpu
- context
- runcpu

## sw subcommand

### Purpose

The **sw** subcommand allows a selected thread to be considered the current thread.

### Syntax

**sw** [ {*th_slot* | *th_Address*} | {**u** | **k**} ]

### Parameters

- **u** – Switches to user address space for the current thread.
- **k** – Switches to kernel address space for the current thread.
- *th_slot* – Specifies a thread slot number. This parameter must be a decimal value.
- *th_Address* – Specifies the address of a thread slot. Symbols, hexadecimal values, or hexadecimal expressions can be used to specify the address.

The **u** and **k** flags can be used to switch between the user and kernel address space for the current thread.

By default, KDB shows the virtual space for the current thread. Threads can be specified by slot number or address. The current thread can be reset to its initial context by entering the **sw** subcommand with no parameters. For the KDB kernel debugger, the initial context is also restored whenever you exit the KDB kernel debugger.

### Aliases

**switch**

### Example

The following is an example of how to use the **sw** subcommand:

```
KDB(0)> sw 12  //switch to thread slot 12
Switch to thread: <thread+000900>
KDB(0)> f  //print stack trace
thread+000900 STACK:
[000215FC]e_block_thread+000250 ()
[00021C48]e_sleep_thread+000070 (??, ??, ??)
[000200F4]errread+00009C (??, ??)
[001C89B4]rdevread+000120 (??, ??, ??, ??)
[0023A61C]cdev_rdwr+00009C (??, ??, ??, ??, ??, ??, ??)
[00216324]spec_rdwr+00008C (??, ??, ??, ??, ??, ??, ??, ??)
[001CEA3C]vnop_rdwr+000070 (??, ??, ??, ??, ??, ??, ??, ??)
[001BDB0C]rwuio+0000CC (??, ??, ??, ??, ??, ??, ??, ??)
[001BDF40]rdwr+000184 (??, ??, ??, ??, ??, ??)
[001BDD68]kreadv+000064 (??, ??, ??, ??)
[000037D8].sys_call+000000 ()
[D0046B68]read+000028 (??, ??, ??)
[1000167C]child+000120 ()
[10001A84]main+0000E4 (??, ??)
[1000014C].__start+00004C ()
KDB(0)> dr sr  //display segment registers
s0  : 00000000  s1  : 007FFFFF  s2  : 00000AB7  s3  : 007FFFFF  s4  : 007FFFFF
s5  : 007FFFFF  s6  : 007FFFFF  s7  : 007FFFFF  s8  : 007FFFFF  s9  : 007FFFFF
s10 : 007FFFFF  s11 : 007FFFFF  s12 : 007FFFFF  s13 : 6000058B  s14 : 00000204
s15 : 60000CBB
KDB(0)> sw u  //switch to user context
KDB(0)> dr sr  //display segment registers
s0  : 60000000  s1  : 600009B1  s2  : 60000AB7  s3  : 007FFFFF  s4  : 007FFFFF
s5  : 007FFFFF  s6  : 007FFFFF  s7  : 007FFFFF  s8  : 007FFFFF  s9  : 007FFFFF
```

```
s10 : 007FFFFF  s11 : 007FFFFF  s12 : 007FFFFF  s13 : 6000058B  s14 : 007FFFFF
s15 : 60000CBB
//Now it is possible to look at user code
//For example, find how read() is called by child()
KDB(0)> dc 1000167C   //print child() code (seg 1 is now valid)
1000167C      bl    <1000A1BC>
KDB(0)> dc 1000A1BC 6  //print child() code
1000A1BC     lwz    r12,244(toc)
1000A1C0     stw    toc,14(stkp)
1000A1C4     lwz    r0,0(r12)
1000A1C8     lwz    toc,4(r12)
1000A1CC   mtctr    r0
1000A1D0   bcctr
...  //find stack pointer of child() routine with 'set 9; f'
[D0046B68]read+000028 (??, ??, ??)
=========================================================================
2FF22B50: 2FF2 2D70  2000 9910  1000 1680  F00F 3130   /.-p .........10
2FF22B60: F00F 1E80  2000 4C54  0000 0003  0000 4503   .... .LT......E.
2FF22B70: 2FF2 2B88  0000 D030  0000 0000  6000 0000   /.+....0....`...
2FF22B80: 6000 09B1  0000 0000  0000 0002  0000 0002   `...............
=========================================================================
[1000167C]child+000120 ()
...
(0)> dw 2FF22B50+14 1       //- stw toc,14(stkp)
2FF22B64: 20004C54          //toc address
(0)> dw 20004C54+244 1      //- lwz r12,244(toc)
20004E98: F00BF5C4          //function descriptor address
(0)> dw F00BF5C4 2          //- lwz r0,0(r12) - lwz toc,4(r12)
F00BF5C4: D0046B40 F00C1E9C //function descriptor (code and toc)
(0)> dc D0046B40 11         //- bcctr will branch to:
D0046B40    mflr    r0
D0046B44     stw    r31,FFFFFFFC(stkp)
D0046B48     stw    r0,8(stkp)
D0046B4C    stwu    stkp,FFFFFFB0(stkp)
D0046B50     stw    r5,3C(stkp)
D0046B54     stw    r4,38(stkp)
D0046B58     stw    r3,40(stkp)
D0046B5C   addic    r4,stkp,38
D0046B60      li    r5,1
D0046B64      li    r6,0
D0046B68      bl    <D00ADC68>  //read+000028
```

The following example shows some of the differences between kernel and user mode for 64-bit process:

```
(0)> sw k      //kernel mode
(0)> dr msr      //kernel machine status register
msr   : 000010B0  bit set: ME  IR DR
(0)> dr r1     //kernel stack pointer
r1  : 2FF3B2A0   2FF3B2A0
(0)> f          //stack frame (kernel MST)
thread+002A98 STACK:
[00031960]e_block_thread+000224 ()
[00041738]nsleep+000124 (??, ??)
[01CFF0F4]nsleep64_+000058 (0FFFFFFF, F0000001, 00000001, 10003730, 1FFFFEF0, 1FFFFEF8)
[000038B4].sys_call+000000 ()
[80000010000867C]080000010000867C (??, ??, ??, ??)
[80000010001137C]nsleep+000094 (??, ??)
[800000100058204]sleep+000030 (??)
[100000478]main+0000CC (0000000100000001, 00000000200FEB78)
[10000023C]__start+000044 ()
(0)> sw u      //user mode
(0)> dr msr      //user machine status register
msr   : 800000004000D0B0  bit set: EE PR ME  IR DR
(0)> dr r1     //user stack pointer
r1  : 0FFFFFFFFFFFFF00   0FFFFFFFFFFFFF00
```

```
(0)> f           //stack frame (kernel MST extension)
thread+002A98 STACK:
[8000001000581D4]sleep+000000 (0000000000000064 [??])
[100000478]main+0000CC (0000000100000001, 00000000200FEB78)
[10000023C]__start+000044 ()
```

# cpu subcommand

## Purpose

The **cpu** subcommand allows you to switch from the current processor to the specified processor.

## Syntax

**cpu** [ **cpu** *number* | **any** ]

## Parameters

- **cpu** *number* – Specifies the CPU number. This value must be a decimal value.
- **any** – Unblocks switched processors.

Without a parameter, the **cpu** subcommand prints processor status.

For the **kdb** command, the processor status displays the address of the Per Processor Data Area (PPDA) for the processor, the current thread for the processor, and the Current Save state Address (CSA).

For the KDB kernel debugger, the processor status indicates the current state of the processor (for example, stopped, switched, debug, and so forth). A switched processor is blocked until the next **start** or **cpu** subcommand. Switching between processors does not change the processor state.

**Note:** If a selected processor cannot be reached, you can go back to the previous processor by typing ^\\ twice.

## Aliases

No aliases.

## Example

The following is an example of how to use the **cpu** subcommand:

```
KDB(4)> cpu  //display processors status
cpu 0 status VALID SWITCHED action SWITCH
cpu 1 status VALID SWITCHED action SWITCH
cpu 2 status VALID SWITCHED action SWITCH
cpu 3 status VALID SWITCHED action SWITCH
cpu 4 status VALID DEBUG action RESUME
cpu 5 status VALID SWITCHED action SWITCH
cpu 6 status VALID SWITCHED action SWITCH
cpu 7 status VALID SWITCHED action SWITCH
KDB(4)> cpu 7  //switch to processor 7
Debugger entered via keyboard.
.waitproc+0000B0     lbz    r0,0(r30)           r0=0,0(r30)=ppda+0014D0
KDB(7)> cpu  //display processors status
cpu 0 status VALID SWITCHED action SWITCH
cpu 1 status VALID SWITCHED action SWITCH
cpu 2 status VALID SWITCHED action SWITCH
cpu 3 status VALID SWITCHED action SWITCH
cpu 4 status VALID SWITCHED action SWITCH
cpu 5 status VALID SWITCHED action SWITCH
cpu 6 status VALID SWITCHED action SWITCH
cpu 7 status VALID DEBUG
KDB(7)>
```

## ctx subcommand

### Purpose

The **ctx** subcommand is used to switch between cpu contexts when viewing a system memory dump.

**Note:** This subcommand is only available within the **kdb** command. It cannot be used with the KDB kernel debugger.

### Syntax

**ctx** [*cpu number*]

### Parameters

- *cpu number* – decimal value or expression indicating a CPU number. If the CPU number is not given as an parameter, the initial context is restored.

**Note:** You can select KDB context to see more information through the stack trace subcommand. For example, you could see a complete stack of a kernel panic. However, KDB context is available only if the running kernel is booted with KDB kernel debugger.

### Aliases

**context**

### Example

The following is an example of how to use the **ctx** subcommand:

```
$ kdb dump unix  //dump analysis
Preserving 628325 bytes of symbol table
First symbol sys_resource
Component Names:
 1)  proc
 2)  thrd
 3)  errlg
 4)  bos
 5)  vmm
 6)  bscsi
 7)  scdisk
 8)  lvm
 9)  tty
10)  netstat
11)  lent_dd

PFT:
id....................0007
raddr.....0000000001000000 eaddr.....0000000001000000
size.............00800000 align.............00800000
valid..1 ros....0 holes..0 io.....0 seg....1 wimg...2

PVT:
id....................0008
raddr.....00000000004B8000 eaddr.....00000000004B8000
size.............000FFD60 align.............00001000
valid..1 ros....0 holes..0 io.....0 seg....1 wimg...2
Dump analysis on POWER_PC POWER_604 machine with 8 cpu(s)
Processing symbol table...
.......................done
(0)> stat     //machine status
RS6K_SMP_MCA POWER_PC POWER_604 machine with 8 cpu(s)
.......... SYSTEM STATUS
sysname... AIX        nodename.. jumbo32
```

```
release... 3          version... 4
machine... 00920312A0 nid....... 920312A0
time of crash: Tue Jul 22 09:46:22 1997
age of system: 1 day, 0 min., 35 sec.
.......... PANIC STRING
assert(v_lookup(sid,pno) == -1)
.......... SYSTEM MESSAGES

AIX 4.3
Starting physical processor #1 as logical #1... done.
Starting physical processor #2 as logical #2... done.
Starting physical processor #3 as logical #3... done.
Starting physical processor #4 as logical #4... done.
Starting physical processor #5 as logical #5... done.
Starting physical processor #6 as logical #6... done.
Starting physical processor #7 as logical #7... done.
[v_lists.c #727]
<- end_of_buffer
(0)> ctx 0     //KDB context of CPU 0
Switch to KDB context of cpu 0
(0)> dr iar     //current instruction
iar   : 00009414
.unlock_enable+000110      lwz    r0,8(stkp)          r0=0,8(stkp)=mststack+00AD18
(0)> ctx 1     //KDB context of CPU 1
Switch to KDB context of cpu 1
(1)> dr iar     //current instruction
iar   : 000BDB68
.kunlockl+000118      blr                        <.ld_usecount+0005BC> r3=0000000B
(1)> ctx 2     //KDB context of CPU 2
Switch to KDB context of cpu 2
(2)> dr iar     //current instruction
iar   : 00027634
.tstart+000284      blr                        <.sys_timer+000964> r3=00000005
(2)> ctx 3     //KDB context of CPU 3
Switch to KDB context of cpu 3
(3)> dr iar     //current instruction
iar   : 01B6A580
01B6A580      ori    r3,r31,0            <00000089> r3=50001000,r31=00000089
(3)> ctx 4     //KDB context of CPU 4
Switch to KDB context of cpu 4
(4)> dr iar     //current instruction
iar   : 00014BFC
.panic_trap+000004       bl    <.panic_dump>       r3=_$STATIC+000294
(4)> f     //current stack
__kdb_thread+0002F0 STACK:
[00014BFC].panic_trap+000004 ()
[0003ACAC]v_inspft+000104 (??, ??, ??)
[00048DA8]v_inherit+0004A0 (??, ??, ??)
[000A7ECC]v_preinherit+000058 (??, ??, ??)
[00027BFC]begbt_603_patch_2+000008 (??, ??)

Machine State Save Area [2FF3B400]
iar   : 00027AEC  msr   : 000010B0  cr    : 22222222  lr    : 00243E58
ctr   : 00000000  xer   : 00000000  mq    : 00000000
r0  : 000A7E74  r1  : 2FF3B220  r2  : 002EBC70  r3  : 00013350  r4  : 00000000
r5  : 00000100  r6  : 00009030  r7  : 2FF3B400  r8  : 00000106  r9  : 00000000
r10 : 00243E58  r11 : 2FF3B400  r12 : 000010B0  r13 : 000C1C80  r14 : 2FF22A88
r15 : 20022DB8  r16 : 20006A98  r17 : 20033128  r18 : 00000000  r19 : 0008AD56
r20 : B02A6038  r21 : 0000006A  r22 : 00000000  r23 : 0000FFFF  r24 : 00000100
r25 : 00003262  r26 : 00000000  r27 : B02B8AEC  r28 : B02A9F70  r29 : 00000001
r30 : 00003350  r31 : 00013350
s0  : 00000000  s1  : 007FFFFF  s2  : 0000864B  s3  : 007FFFFF  s4  : 007FFFFF
s5  : 007FFFFF  s6  : 007FFFFF  s7  : 007FFFFF  s8  : 007FFFFF  s9  : 007FFFFF
s10 : 007FFFFF  s11 : 00001001  s12 : 00002002  s13 : 6001F01F  s14 : 00004004
s15 : 007FFFFF
prev      00000000 kjmpbuf   00000000 stackfix  00000000 intpri    0B
curid     0008AD56 sralloc   E01E0000 ioalloc   00000000 backt     00
```

```
flags     00 tid       00000000 excp_type 00000000
fpscr     00000000 fpeu            01 fpinfo          00 fpscrx    00000000
o_iar     00000000 o_toc     00000000 o_arg1    00000000
excbranch 00000000 o_vaddr   00000000 mstext    00000000
Except :
 csr   00000000 dsisr 40000000  bit set: DSISR_PFT
 srval 6000864B dar   2FF22FF8 dsirr 00000106

[00027AEC].backt+000000 (00013350, 00000000 [??])
[00243E54]vms_delete+0004DC (??)
[00256838]shmfreews+0000B0 ()
[000732B4]freeuspace+000010 ()
[00072EAC]kexitx+000688 (??)
(4)> ctx    //AIX context of CPU 4
Restore initial context
(4)> f     //current stack
thread+031920 STACK:
[00027AEC].backt+000000 (00013350, 00000000 [??])
[00243E54]vms_delete+0004DC (??)
[00256838]shmfreews+0000B0 ()
[000732B4]freeuspace+000010 ()
[00072EAC]kexitx+000688 (??)
(4)>
```

## runcpu subcommand

### Purpose

The **runcpu** subcommand allows you to run any other **kdb** subcommand to for every processor in the system. It is intended for use with subcommands such as the **f** subcommand for which the output depends on the current processor in the KDB kernel debugger.

### Syntax

**runcpu** *cmd*

### Parameters

- *cmd* – Specifies the kdb subcommand that is to be run for every processor in the system.

The specified command only runs on processors that the KDB kernel debugger has stopped. If errors occur when the command is run on a particular processor, the **runcpu** subcommand continues and runs the command on the next processor. The **runcpu** subcommand can be stopped by pressing Ctrl+C.

### Aliases

No aliases.

### Example

The following is an example of how to use the **runcpu** subcommand:

```
KDB(0)> runcpu f

--- CPU #0 ---
pvthread+000200 STACK:
[00026078]waitproc_find_run_queue+00018C (0000000000000001 [??])
[000285DC]waitproc+000134 ()
[000DE8F8]procentry+000010 (??, ??, ??, ??)

--- CPU #1 ---
pvthread+000300 STACK:
[00026124]waitproc_find_run_queue+000238 (0000000000000080 [??])
[000285DC]waitproc+000134 ()
[000DE8F8]procentry+000010 (??, ??, ??, ??)
KDB(0)>
```

# Chapter 11. Calculator and converter subcommands

The subcommands in this category are used to convert decimal numbers to other formats and evaluate decimal and hexadecimal expressions. These subcommands include the following:

- hcal
- dcal
- conv

# hcal and dcal subcommands

## Purpose

The **hcal** subcommand evaluates hexadecimal expressions and displays the result in both hexadecimal and decimal. The **dcal** subcommand evaluates decimal expressions and displays the result in both hexadecimal and decimal.

## Syntax

**hcal** *HexadecimalExpression*

**dcal** *DecimalExpression*

## Parameters

- *HexadecimalExpression* – Specifies the hexadecimal expression to be evaluated.
- *DecimalExpression* – Specifies the decimal expression to be evaluated.

## Aliases

**hcal** – **cal**

**dcal** has no alias.

## Example

The following is an example of how to use the **dcal** subcommand and the **hcal** subcommand:

```
KDB(0)> hcal 0x10000  //convert a single value
Value hexa: 00010000          Value decimal: 65536
KDB(0)> dcal 1024*1024  //convert an expression
Value decimal: 1048576          Value hexa: 00100000
KDB(0)> set 11  //64 bits printing
64_bit is true
KDB(0)> hcal 0-1  //convert -1
Value hexa: FFFFFFFFFFFFFFFF  Value decimal: -1 Unsigned: 18446744073709551615
KDB(0)> set 11  //32 bits printing
64_bit is false
KDB(0)> hcal 0-1  //convert -1
Value hexa: FFFFFFFF          Value decimal: -1 Unsigned: 4294967295
```

## conv subcommand

### Purpose

The **conv** subcommand converts an arbitrary base number to a decimal, binary, octal, or hexadecimal number.

### Syntax

**conv** [ **-b** | **-d** | **-o** | **-x** | **-a** *base* ] [ **-s** ] *value*

### Parameters

- **-b** – Specifies that the number to convert specified by the *value* parameter is a binary number.
- **-d** – Specifies that the number to convert specified by the *value* parameter is a decimal number.
- **-o** – Specifies that the number to convert specified by the *value* parameter is an octal number.
- **-x** – Specifies that the number to convert specified by the *value* parameter is a hexadecimal number.
- **-a** *base* – Specifies that the number to convert specified by the *value* parameter is a number with the arbitrary base of *base*. The number must be between 2 and 36 inclusive.
- **-s** – Extends the left-most, one-bit sign of the number to convert specified by the *value* parameter.
- *value* – Specifies the number to convert.

### Aliases

No aliases.

### Example

The following is an example of how to use the **conv** subcommand:

```
KDB(0)> conv 1101
Binary : 0000000000000000000000000000000000000000000000000000010001001101
Octal  : 0000000000000000002115
Decimal: 1101
Hex    : 000000000000044D
KDB(0)> conv -b 1101
Binary : 0000000000000000000000000000000000000000000000000000000000001101
Octal  : 0000000000000000000015
Decimal: 13
Hex    : 000000000000000D
KDB(0)> conv -b -s 1101
Binary : 1111111111111111111111111111111111111111111111111111111111111101
Octal  : 1777777777777777777775
Decimal: -3
Hex    : FFFFFFFFFFFFFFFD
KDB(0)>
```

# Chapter 12. CPU start and stop subcommands

The subcommands in this category are used to selectively hold processors in kdb spin loops and then release them back to general operating system use. These subcommands include the following:

- start
- stop

# start and stop subcommands

## Purpose

The **start** subcommand starts all processors or a specific processor. The **stop** subcommand stops all processors or a specific processor.

**Note:** These subcommands are only available within the KDB kernel debugger. They are not included in the **kdb** command.

## Syntax

**start** *cpu_number* | **all**

**stop** *cpu_number* | **all**

## Parameters

- *cpu_number* – Specifies the CPU number to start or stop. This parameter must be a decimal value.
- **all** – Indicates that all processors are to be started or stopped.

When a processor is stopped, it is looping inside the KDB kernel debugger and the processor does not go back to the operating system.

## Aliases

No aliases.

## Example

The following is an example of how to use the **start** subcommand and the **stop** subcommand:

```
KDB(1)> stop 0  //stop processor 0
KDB(1)> cpu  //display processors status
cpu 0 status VALID STOPPED action STOP
cpu 1 status VALID DEBUG
KDB(1)> start 0  //start processor 0
KDB(1)> cpu  //display processors status
cpu 0 status VALID action START
cpu 1 status VALID DEBUG
KDB(1)> b sy_decint  //set break point
KDB(1)> e  //exit the debugger
Breakpoint
.sy_decint+000000    mflr    r0                 <.dec_flih+000014>
KDB(0)> cpu  //display processors status
cpu 0 status VALID DEBUG action RESUME
cpu 1 status VALID DEBUGWAITING
KDB(0)> cpu 1  //switch to processor 1
Breakpoint
.sy_decint+000000    mflr    r0                 <.dec_flih+000014>
KDB(1)> cpu  //display processors status
cpu 0 status VALID SWITCHED action SWITCH
cpu 1 status VALID DEBUG
KDB(1)> cpu 0  //switch to processor 0
KDB(0)> cpu  //display processors status
cpu 0 status VALID DEBUG
cpu 1 status VALID SWITCHED action SWITCH
KDB(0)> q  //exit the debugger
```

# Chapter 13. Basic display subcommands

The subcommands in this category display stack frames, system statistics and information about processors. These subcommands include the following:

- f
- status
- stat
- pr
- symptom

## f subcommand

### Purpose

The **f** subcommand displays all of the stack frames from the current instruction as deep as possible. Interrupts and system calls are crossed and the user stack is displayed.

### Syntax

**f** [**+x** | **-x**] [**th** {*slot* | *address*} ]

### Parameters

- **+x** – Includes hexadecimal addresses as well as symbolic names for calls on the stack. This option remains set for future invocations of the stack subcommand until it is changed using the **-x** flag.
- **-x** – Suppresses the display of hexadecimal addresses for functions on the stack. This option remains in effect for future invocations of the stack subcommand until it is changed using the **+x** flag.
- *slot* – Indicates the thread slot number. It is a decimal value
- *Address* – Indicates the effective address for a thread slot. It is a hexadecimal address, hexadecimal expression, or symbol.

In the user space, trace back allows the display of symbolic names, but the KDB kernel debugger cannot directly access these symbols. Use the **+x** toggle to have hexadecimal addresses displayed (for example, to put a break point on one of these addresses). If invoked with no parameter, the stack for the current thread is displayed. The stack for a particular thread can be displayed by specifying its slot number or address.

**Note:** The amount of data displayed can be controlled through the **mst_wanted** and **display_stack_wanted** options of the **set** subcommand. For more information, see "set subcommand" on page 44.

For some compilation options, specifically **-O**, routine parameters are not saved in the stack. KDB warns about this by displaying [**??**] at the end of the line. In this case, the displayed routine parameters might be wrong.

### Aliases

**stack**, **where**

### Example

The following is an example of how to use the **f** subcommand. In the following example, a break point is set on **v_gettlock** and when the break point is encountered, the stack is displayed. The first parameter of the **open()** syscall is displayed and saved by **copen()** in register R31. Register R31 is saved in the stack by **openpath()**. The first parameter is found by looking at the memory pointed to by register R31.

```
KDB(2)> f  //show the stack
thread+012540 STACK:
[0004AC84]v_gettlock+000000 (00012049, C0011E80, 00000080, 00000000 [??])  <-- Optimized code, note [??]
[00085C18]v_pregettlock+0000B4 (??, ??, ??, ??)
[000132E8]isync_vcs1+0000D8 (??, ??)
____ Exception (2FF3B400) ____
[000131FC].backt+000000 (00012049, C0011E80 [??])  <-- Optimized code, note [??]
[0004B220]vm_gettlock+000020 (??, ??)
[0019A64C]iwrite+00013C (??)
[0019D194]finicom+0000A0 (??, ??)
[0019D4F0]comlist+0001CC (??, ??)
[0019D5BC]_commit+000030 (00000000, 00000001, 09C6E9E8, 399028AA,
0000A46F, 0000E2AA, 2D3A4EAA, 2FF3A730)
[001E1B18]jfs_setattr+000258 (??, ??, ??, ??, ??, ??)
```

```
[001A5ED4]vnop_setattr+000018 (??, ??, ??, ??, ??, ??)
[001E9008]spec_setattr+00017C (??, ??, ??, ??, ??, ??)
[001A5ED4]vnop_setattr+000018 (??, ??, ??, ??, ??, ??)
[01B655C8]pty_vsetattr+00002C (??, ??, ??, ??, ??, ??)
[01B6584C]pty_setname+000084 (??, ??, ??, ??, ??, ??)
[01B60810]pty_create_ptp+0002C4 (??, ??, ??, ??, ??)
[01B60210]pty_open_comm+00015C (??, ??, ??, ??)
[01B5FFC0]call_pty_open_comm+0000B8 (??, ??, ??, ??)
[01B6526C]ptm_open+000140 (??, ??, ??, ??, ??)
(2)> more (^C to quit) ?
[01A9A124]open_wrapper+0000D0 (??)
[01A8DF74]csq_protect+000258 (??, ??, ??, ??, ??, ??)
[01A96348]osr_open+0000BC (??)
[01A9C1C8]pse_clone_open+000164 (??, ??, ??, ??)
[001ADCC8]spec_clone+000178 (??, ??, ??, ??, ??)
[001B3FC4]openpnp+0003AC (??, ??, ??, ??, ??)
[001B4178]openpath+000064 (??, ??, ??, ??, ??, ??)
[001B43E8]copen+000130 (??, ??, ??, ??, ??)
[001B44BC]open+000014 (??, ??, ??)
[000037D8].sys_call+000000 ()
[10002E74]doit+00003C (??, ??, ??)
[10003924]main+0004CC (??, ??)
[1000014C].__start+00004C ()
KDB(2)> set 10  //show saved registers
display_stacked_regs is true
KDB(2)> f  //show the stack
thread+012540 STACK:
[0004AC84]v_gettlock+000000 (00012049, C0011E80, 00000080, 00000000 [??])
...
[001B3FC4]openpnp+0003AC (??, ??, ??, ??, ??)
r24 : 2FF3B6E0 r25 : 2FF3B400 r26 : 10002E78 r27 : 00000000 r28 : 00000002
r29 : 2FF3B3C0 r30 : 00000000 r31 : 20000510
[001B4178]openpath+000064 (??, ??, ??, ??, ??, ??)
[001B43E8]copen+000130 (??, ??, ??, ??, ??)
r27 : 2A22A424 r28 : E3014000 r29 : E6012540 r30 : 0C87B000 r31 : 00000000
[001B44BC]open+000014 (??, ??, ??)
...
KDB(2)> dc open 6  //look for parameter R3
.open+000000    stwu    stkp,FFFFFFC0(stkp)
.open+000004    mflr    r0
.open+000008   addic    r7,stkp,38
.open+00000C     stw    r0,48(stkp)
.open+000010      li    r6,0
.open+000014      bl    <.copen>
KDB(2)> dc copen 9  //look for parameter R3
.copen+000000    stmw    r27,FFFFFFEC(stkp)
.copen+000004    addi    r28,r4,0
.copen+000008    mflr    r0
.copen+00000C     lwz    r4,D5C(toc)         D5C(toc)=audit_flag
.copen+000010     stw    r0,8(stkp)
.copen+000014    stwu    stkp,FFFFFFA0(stkp)
.copen+000018    cmpi    cr0,r4,0
.copen+00001C   mtcrf    cr5,r28
.copen+000020    addi    r31,r3,0
KDB(2)> d 20000510  //display memory location @R31
20000510: 2F64 6576  2F70 7463  0000 0000  416C 6C20   /dev/ptc....All
```

In the following example, you must find what the **lsfs** subcommand is waiting for. The answer is given with **getfssize** parameters, which are saved in the stack.

```
# ps -ef|grep lsfs
 root  63046  39258   0   Apr 01  pts/1  0:00 lsfs
# kdb
Preserving 587377 bytes of symbol table
First symbol sys_resource
PFT:
id....................0007
raddr.............01000000 eaddr.............B0000000
```

```
size.............01000000 align............01000000
valid..1 ros....0 holes..0 io.....0 seg....0 wimg...2

PVT:
id....................0008
raddr............003BC000 eaddr............B2000000
size.............001FFDA0 align............00001000
valid..1 ros....0 holes..0 io.....0 seg....0 wimg...2
(0)> dcal 63046  //print hexadecimal value of PID
Value decimal: 63046          Value hexa: 0000F646
(0)> tpid 0000F646  //show threads of this PID
           SLOT NAME     STATE   TID PRI CPUID CPU FLAGS    WCHAN

thread+025440  795 lsfs     SLEEP 31B31 03C       000 00000004 057DB5BC
(0)> sw 795  //set current context on this thread
Switch to thread: <thread+025440>
(0)> f  //show the stack
thread+025440 STACK:
[000205C0]e_block_thread+000250 ()
[00020B1C]e_sleep_thread+000040 (??, ??, ??)
[0002AAA0]iowait+00004C (??)
[0002B40C]bread+0000DC (??, ??)
[0020AF4C]readblk+0000AC (??, ??, ??, ??)
[001E90D8]spec_rdwr+00007C (??, ??, ??, ??, ??, ??, ??, ??)
[001A6328]vnop_rdwr+000070 (??, ??, ??, ??, ??, ??, ??, ??)
[00198278]rwuio+0000CC (??, ??, ??, ??, ??, ??, ??, ??)
[001986AC]rdwr+000184 (??, ??, ??, ??, ??, ??)
[001984D4]kreadv+000064 (??, ??, ??, ??)
[000037D8].sys_call+000000 ()
[D0046A18]read+000028 (??, ??, ??)
[1000A0E4]get_superblk+000054 (??, ??, ??)
[100035F8]read_super+000024 (??, ??, ??, ??)
[10005C00]getfssize+0000A0 (??, ??, ??)
[10002D18]prnt_stanza+0001E8 (??, ??, ??)
[1000349C]do_ls+000294 (??, ??)
[10000524]main+0001E8 (??, ??)
[1000014C].__start+00004C ()
(0)> sw u  //enable user context of the thread
(0)> dc 10005C00-a0 8  //look for parameters R3, R4, R5
10005B60    mflr    r0
10005B64     stw    r31,FFFFFFFC(stkp)
10005B68     stw    r0,8(stkp)
10005B6C    stwu    stkp,FFFFFEE0(stkp)
10005B70     stw    r3,108(stkp)
10005B74     stw    r4,104(stkp)
10005B78     stw    r5,10C(stkp)
10005B7C    addi    r3,r4,0
(0)> set 9  //print stack frame
display_stack_frames is true
(0)> f  //show the stack
thread+025440 STACK:
[000205C0]e_block_thread+000250 ()
...
[100035F8]read_super+000024 (??, ??, ??, ??)
=====================================================================
2FF225D0: 2FF2 26F0  2A20 2429  1000 5C04  F071 71C0   /.&.* $)..\..qq.
2FF225E0: 2FF2 2620  2000 4D74  D000 4E18  F071 F83C   /.&  .Mt..N..q.<
2FF225F0: F075 2FF8  F074 36A4  F075 0FE0  F075 1FF8   .u/..t6..u...u..
2FF22600: F071 AE80  8080 8080  0000 0004  0000 0006   .q..............
=====================================================================
[10005C00]getfssize+0000A0 (??, ??, ??)
...
(0)> dw 2FF225D0+104  //print parameters (offset 0x104 0x108 0x10c)
2FF226D4: 2000DCC8 2000DC78 00000000 00000004
(0)> d 2000DC78 20  //print first parameter
2000DC78: 2F74 6D70  2F73 7472  6970 655F  6673 2E32   /tmp/stripe_fs.2
2000DC88: 3433 3632  0000 0000  0000 0000  0000 0004   4362............
```

```
(0)> d 2000DCC8 20  //print second parameter
2000DCC8: 2F64 6576  2F73 6C76  3234 3336  3200 0000   /dev/slv24362...
2000DCD8: 0000 0000  0000 0000  0000 0000  0000 0004   ................
(0)> q  //leave debugger
#
```

## status subcommand

### Purpose

The **status** subcommand displays information about what is currently running on each processor.

### Syntax

**status** [*cpu*]

### Parameters

- *cpu* – Specifies the CPU number.

If no argument is specified, information is displayed for all processors.

### Aliases

No aliases.

### Example

The following is an example of how to use the **status** subcommand:

```
KDB(0)> status
CPU     TID  TSLOT     PID  PSLOT  PROC_NAME
  0      205     2      204     2  wait
  1      307     3      306     3  wait
KDB(0)> status 1
CPU     TID  TSLOT     PID  PSLOT  PROC_NAME
  1      307     3      306     3  wait
```

## stat subcommand

### Purpose

The **stat** subcommand displays system statistics that include the last kernel **printf()** messages still in memory.

### Syntax

**stat**

### Parameters

No parameters.

The following information is displayed for a processor that has crashed:

- Processor logical number
- Current Save Area (CSA) address
- LED value

For the KDB kernel debugger, this subcommand also displays the reason why the debugger was entered. There is one reason per processor.

### Aliases

No aliases.

### Example

The following is an example of how to use the **stat** subcommand:

```
KDB(6)> stat  //machine status using the KDB kernel debugger
RS6K_SMP_MCA POWER_PC POWER_604 machine with 8 cpu(s)
SYSTEM STATUS:
sysname: AIX
nodename: jumbo32
release: 2
version: 4
machine: 00920312A000
nid: 920312A0
Illegal Trap Instruction Interrupt in Kernel
age of system: 1 day, 5 hr., 59 min., 50 sec.

SYSTEM MESSAGES

AIX 4.2
Starting physical processor #1 as logical #1... done.
Starting physical processor #2 as logical #2... done.
Starting physical processor #3 as logical #3... done.
Starting physical processor #4 as logical #4... done.
Starting physical processor #5 as logical #5... done.
Starting physical processor #6 as logical #6... done.
Starting physical processor #7 as logical #7... done.
<- end_of_buffer
CPU 6 CSA 00427EB0 at time of crash, error code for LEDs: 70000000

(0)> stat  //machine status using the kdb command running on the dump file
RS6K_SMP_MCA POWER_PC POWER_604 machine with 4 cpu(s)
.......... SYSTEM STATUS
sysname... AIX        nodename.. zoo22
release... 3          version... 4
machine... 00989903A6 nid....... 989903A6
time of crash: Sat Jul 12 12:34:32 1997
```

```
age of system: 1 day, 2 hr., 3 min., 49 sec.
.......... SYSTEM MESSAGES

AIX 4.3
Starting physical processor #1 as logical #1... done.
Starting physical processor #2 as logical #2... done.
Starting physical processor #3 as logical #3... done.
<- end_of_buffer
.......... CPU 0 CSA 004ADEB0 at time of crash, error code for LEDs: 30000000
thread+01B438 STACK:
[00057F64]v_sync+0000E4 (B01C876C, 0000001F [??])
[000A4FA0]v_presync+000050 (??, ??)
[0002B05C]begbt_603_patch_2+000008 (??, ??)

Machine State Save Area [2FF3B400]
iar   : 0002AF4C  msr   : 000010B0  cr    : 24224220  lr    : 0023D474
ctr   : 00000004  xer   : 20000008  mq    : 00000000
r0  : 000A4F50  r1  : 2FF3A600  r2  : 002E62B8  r3  : 00000000  r4  : 07D17B60
r5  : E601B438  r6  : 00025225  r7  : 00025225  r8  : 00000106  r9  : 00000004
r10 : 0023D474  r11 : 2FF3B400  r12 : 000010B0  r13 : 000C0040  r14 : 2FF229A0
r15 : 2FF229BC  r16 : DEADBEEF  r17 : DEADBEEF  r18 : DEADBEEF  r19 : 00000000
r20 : 0048D4C0  r21 : 0048D3E0  r22 : 07D6EE90  r23 : 00000140  r24 : 07D61360
r25 : 00000148  r26 : 0000014C  r27 : 07C75FF0  r28 : 07C75FFC  r29 : 07C75FF0
r30 : 07D17B60  r31 : 07C76000
s0  : 00000000  s1  : 007FFFFF  s2  : 00001DD8  s3  : 007FFFFF  s4  : 007FFFFF
s5  : 007FFFFF  s6  : 007FFFFF  s7  : 007FFFFF  s8  : 007FFFFF  s9  : 007FFFFF
s10 : 007FFFFF  s11 : 00000101  s12 : 0000135B  s13 : 00000CC5  s14 : 00000404
s15 : 6000096E
prev      00000000 kjmpbuf   2FF3A700 stackfix  00000000 intpri    0B
curid     00003C60 sralloc   E01E0000 ioalloc   00000000 backt     00
flags     00 tid       00000000 excp_type 00000000
fpscr     00000000 fpeu            00 fpinfo          00 fpscrx    00000000
o_iar     00000000 o_toc     00000000 o_arg1    00000000
excbranch 00000000 o_vaddr   00000000 mstext    00000000
Except :
 csr   00000000 dsisr 40000000  bit set: DSISR_PFT
 srval 00000000 dar   07CA705C dsirr 00000106

[0002AF4C].backt+000000 (00000000, 07D17B60 [??])
[0023D470]ilogsync+00014C (??)
[002894B8]logsync+000090 (??)
[0028899C]logmvc+000124 (??, ??, ??, ??)
[0023AB68]logafter+000100 (??, ??, ??)
[0023A46C]commit2+0001EC (??)
[0023BF50]finicom+0000BC (??, ??)
[0023C2CC]comlist+0001F0 (??, ??)
[0029391C]jfs_rename+000794 (??, ??, ??, ??, ??, ??, ??)
[00248220]vnop_rename+000038 (??, ??, ??, ??, ??, ??, ??)
[0026A168]rename+000380 (??, ??)
(0)>
```

## pr subcommand

### Purpose

The **pr** subcommand displays memory as if it were of a specified type (c data structure).

### Syntax

**pr** [*type*] *address*

**pr -l** *offset* |*name* [**–e** *end_val*] [*type*] *address*

**pr -a** *count* [*type*] *address*

**pr -d** *default_type*

**pr -p** *pattern*

### Parameters

- **-l** – Displays data following a linked list. The **pr** subcommand follows the linked list until the value in the linked list pointer equals the ending value. The ending value is zero, unless it is changed with the **-e** parameter.
- **-e** – Changes the ending value used when you are displaying a linked list.
- **-a** – Displays the data as if it were an array whose elements are of the specified type.
- **-d** – Sets the default type.
- *default_type* – Indicates the type (c data structure) for which you want to display information. After you set the default type by using the –d parameter, it is the only type for which information is displayed.
- **-p** – Displays the defined symbols that match a specified pattern.
- *type* – Specifies the type used to display the data.
- *address* – Specifies the effective address of the data to be displayed.
- *offset* – Specifies the offset of the linked list pointer in the data structure.
- *name* – Specifies the name of the linked list pointer in the data structure.
- *end_val* – Specifies the new ending value.
- *count* – Specifies the number of elements to display.
- *pattern* – Specifies the pattern.

Before a *type* can be used, it must be loaded into the kernel with the **bosdebug -l** command. The **bosdebug** command must be issued outside of **kdb** as the root user. It is not necessary to reboot the machine after running the **bosdebug** command.

### Aliases

**print**

### Example

The following is an example of how to use the **pr** subcommand:

```
KDB(0)> pr integer 3000              //use 'pr' without loading symbols
type definition not found

//Run the following as 'root' to load the symbols in intr.h into the kernel
# echo "#include <sys/intr.h>" >sym.c     //symbol file to load into kernel
# echo "main() { }" >>sym.c
# cc -g -o sym sym.c -qdbxextra          //for 32-bit kernel
# cc -g -q64 -o sym sym.c -qdbxextra     //for 64-bit kernel
```

```
# bosdebug -l sym               (load symbols into kernel)
Symbol table initialized. Loaded 297 symbols.

KDB(0)> pr integer 3000               //print data at 0x3000 as an integer
integer foo[0]  = 0x4C696365;
KDB(0)> intr 19                    //show interrupt handler table, slot 19
              SLT INTRADDR HANDLER  TYPE LEVEL    PRIO BID     FLAGS

i_data+00004C  19 30047A80 00000000 0004 00000001 0000 900100C0 0040
i_data+00004C  19 0200C360 0200A908 0004 00000003 0000 900100C0 0040
i_data+00004C  19 319A9020 02041AB8 0004 00000003 0000 900100C0 0040
KDB(0)> intr 30047A80               //show interrupt handler information at 0x30047A80
addr........... 30047A80 handler........ 00000000
bid............ 900100C0 bus_type....... 00000004 BID
next........... 0200C360 flags.......... 00000040 LEVEL
level.......... 00000001 priority....... 00000000 INTMAX
i_count........ 00000000
KDB(0)> pr intr 30047A80          //print this data as an 'intr' structure
struct intr {
    struct intr *next   = 0x0200C360;
    int (*handler)()    = 0x00000000;
    unsigned short bus_type     = 0x0004;
    unsigned short flags        = 0x0040;
    int level   = 0x00000001;
    int priority        = 0x00000000;
    ulong32int64_t bid  = 0x900100C0;
    unsigned long i_count       = 0x00000000;
} foo[0];
KDB(0)> pr 30047A80               //print data using default type
char foo[0]     = 0x02 '';
KDB(0)> pr -d intr                //change default type to 'intr' structure
KDB(0)> pr 30047A80               //print data using new default type
struct intr {
    struct intr *next   = 0x0200C360;
    int (*handler)()    = 0x00000000;
    unsigned short bus_type     = 0x0004;
    unsigned short flags        = 0x0040;
    int level   = 0x00000001;
    int priority        = 0x00000000;
    ulong32int64_t bid  = 0x900100C0;
    unsigned long i_count       = 0x00000000;
} foo[0];
KDB(0)> pr -l next intr 30047A80     //print following the 'next' pointer
struct intr {
    struct intr *next   = 0x0200C360;
    int (*handler)()    = 0x00000000;
    unsigned short bus_type     = 0x0004;
    unsigned short flags        = 0x0040;
    int level   = 0x00000001;
    int priority        = 0x00000000;
    ulong32int64_t bid  = 0x900100C0;
    unsigned long i_count       = 0x00000000;
} foo;
struct intr {
    struct intr *next   = 0x319A9020;
    int (*handler)()    = 0x0200A908;
    unsigned short bus_type     = 0x0004;
    unsigned short flags        = 0x0040;
    int level   = 0x00000003;
    int priority        = 0x00000000;
    ulong32int64_t bid  = 0x900100C0;
    unsigned long i_count       = 0x00000000;
} foo;
struct intr {
    struct intr *next   = 0x00000000;
    int (*handler)()    = 0x02041AB8;
    unsigned short bus_type     = 0x0004;
```

```
    unsigned short flags        = 0x0040;
    int level   = 0x00000003;
    int priority        = 0x00000000;
    ulong32int64_t bid  = 0x900100C0;
    unsigned long i_count       = 0x00000000;
} foo;
KDB(0)> pr -e 319A9020 -l next intr 30047A80     //print following the 'next' pointer,
                                                 //ending when 'next' equals 0x319A9020
struct intr {
    struct intr *next   = 0x0200C360;
    int (*handler)()    = 0x00000000;
    unsigned short bus_type     = 0x0004;
    unsigned short flags        = 0x0040;
    int level   = 0x00000001;
    int priority        = 0x00000000;
    ulong32int64_t bid  = 0x900100C0;
    unsigned long i_count       = 0x00000000;
} foo;
struct intr {
    struct intr *next   = 0x319A9020;
    int (*handler)()    = 0x0200A908;
    unsigned short bus_type     = 0x0004;
    unsigned short flags        = 0x0040;
    int level   = 0x00000003;
    int priority        = 0x00000000;
    ulong32int64_t bid  = 0x900100C0;
    unsigned long i_count       = 0x00000000;
} foo;
KDB(0)> pr -a 2 intr 30047A80          //print two 'intr' stuctures starting at 0x30047A80
struct intr {
    struct intr *next   = 0x0200C360;
    int (*handler)()    = 0x00000000;
    unsigned short bus_type     = 0x0004;
    unsigned short flags        = 0x0040;
    int level   = 0x00000001;
    int priority        = 0x00000000;
    ulong32int64_t bid  = 0x900100C0;
    unsigned long i_count       = 0x00000000;
} foo[0];
struct intr {
    struct intr *next   = 0x00000000;
    int (*handler)()    = 0x00000000;
    unsigned short bus_type     = 0x0000;
    unsigned short flags        = 0x0000;
    int level   = 0x00000000;
    int priority        = 0x00000000;
    ulong32int64_t bid  = 0x00000000;
    unsigned long i_count       = 0x00000000;
} foo[1];
KDB(0)> pr -p intr               //show symbol 'intr'
     intr
KDB(0)> pr -p *r                 //show symbols matching '*r'
     char
     unsigned char
     signed char
     integer
     character
     wchar
     __default_char
     intr
     u_char
     physadr
     uchar
     UTF32Char
     UniChar
KDB(0)> g
```

```
# bosdebug -f                    //unload symbols from kernel
Flushed out all the symbols.

KDB(0)> pr integer 3000               //print after symbols unloaded
type definition not found
```

## symptom subcommand

### Purpose

The **symptom** subcommand displays the symptom string for a dump.

**Note:** The **symptom** subcommand is only available in the **kdb** command.

### Syntax

**symptom** [**-e**]

### Parameters

- **-e** – Writes the symptom string and the stack trace to the system errlog. The symptom string is displayed on the standard output.

If no parameters are used, the **symptom** subcommand displays the symptom string on the standard output.

The **symptom** subcommand is not valid on a running system. The optional **-e** flag creates an error log entry that contains the symptom string. This flag is normally only used by the system and not entered manually. The symptom string can be used to identify duplicate problems.

### Aliases

No aliases.

### Example

- The following example demonstrates the **symptom** command running on a dump:

```
<0> symptom
PIDS/5765C3403 LVLS/430 PCSS/SPI1 MS/300 FLDS/uiocopyin VALU/7ce621ae
FLDS/uiomove VALU/13c
```

- The following example demonstrates the **symptom** subcommand with the **-e** flag running on a dump:

```
<0> symptom -e
PIDS/5765C3403 LVLS/430 PCSS/SPI1 MS/300 FLDS/uiocopyin VALU/7ce621ae
FLDS/uiomove VALU/13c
```

- The corresponding system errlog entry is similar to the following:

```
LABEL:          SYSDUMP_SYMP
   ....
   Detail Data
   DUMP STATUS
   LED:300
   csa:2ff3b400
   uiocopyin_ppc 1c4
   uiomove 13c
   ....
```

# Chapter 14. Memory register display and decode subcommands

The subcommands in this category are used to display and decode the memory register. These subcommands include the following:

- d
- dw
- dd
- dp
- dpw
- dpd
- dc
- dpc
- di
- dr
- ddvb
- ddvh
- ddvw
- ddvd
- ddpb
- ddph
- ddpw
- ddpd

## d, dw, dd, dp, dpw, and dpd subcommands

### Purpose

The **d** (display bytes), **dw** (display words), and **dd** (display double words), subcommands dump memory areas starting at a specified effective address. Access is done in real mode.

The **dp** (display bytes), **dpw** (display words), and **dpd** (display double words) subcommands dump memory areas starting at a specified real address.

### Syntax

**d** *symbol* | *EffectiveAddress* [*count*]

**dw** *symbol* | *EffectiveAddress* [*count*]

**dd** *symbol* | *EffectiveAddress* [*count*]

**dp** *symbol* | *PhysicalAddress* [*count*]

**dpw** *symbol* | *PhysicalAddress* [*count*]

**dpd** *symbol* | *PhysicalAddress* [*count*]

### Parameters

- *EffectiveAddress* – Specifies the virtual (effective) address of the area to be dumped when the **d**, **dw**, or **dd** subcommands are used. Symbols, hexadecimal values, or hexadecimal expressions can be used in specification of the address.
- *PhysicalAddress* – Specifies the physical address of the area to be dumped when the **dp**, **dpw** or **dpd** subcommands are used. Symbols, hexadecimal values, or hexadecimal expressions can be used in specification of the address.
- *count* – Specifies the number of bytes, words, or double words to display. This is a hexadecimal value. The number of bytes are displayed if the **d** subcommand or the **dp** subcommand are used. The number of words are displayed if the **dw** or **dpw** subcommand are used. The number of double words is displayed if the **dd** subcommand or the **dpd** subcommand are used. If no count is specified, 16 bytes of data are displayed.

Any of the display subcommands can be continued from the last address displayed by using the Enter key.

### Aliases

**d** – **dump**

### Example

The following is an example of how to use the **d**, **dw**, **dd**, **dp**, **dpw**, and **dpd** subcommands:

```
KDB(0)> d utsname //display data at utsname
utsname+000000: 4149 5800  0000 0000  0000 0000  0000 0000   AIX.............
KDB(0)> d utsname 8 //display 8 bytes of data at utsname
utsname+000000: 4149 5800  0000 0000                         AIX.....
KDB(0)>   //'enter key' to display the next 8 bytes of data
utsname+000008: 0000 0000  0000 0000                         ........
KDB(0)> dw utsname 8 //display 8 words of data at utsname
utsname+000000: 41495800 00000000 00000000 00000000  AIX.............
utsname+000010: 00000000 00000000 00000000 00000000  ................
KDB(0)> dd utsname 8 //display 8 double-words of data at utsname
utsname+000000: 4149580000000000 0000000000000000  AIX.............
utsname+000010: 0000000000000000 0000000000000000  ................
```

```
utsname+000020: 3030303030303030 4130303000000000  00000000A000....
utsname+000030: 0000000000000000 0000000000000000  ................
KDB(0)> tr utsname //find physical address of utsname
Physical Address = 00000000003D2860
KDB(0)> dp 3D2860 //display data using physical address
003D2860: 4149 5800  0000 0000  0000 0000  0000 0000      AIX.............
KDB(0)> dpw 3D2860 //display data as words using physical address
003D2860: 41495800 00000000 00000000 00000000     AIX.............
KDB(0)> dpd 3D2860 //display data as double-words using physical address
003D2860: 4149580000000000 0000000000000000     AIX.............
KDB(0)>
```

# dc and dpc subcommands

## Purpose

The **dc** and **dpc** subcommands decode instructions.

## Syntax

**dc** *effectiveaddress* [*count*]

**dpc** *physicaladdress* [*count*]

## Parameters

*effectiveaddress* – Specifies the effective or virtual address of the code to disassemble. Symbols, hexadecimal values, or hexadecimal expressions can be used in specification of the address.

*physicaladdress* – Specifies the physical or real address of the code to disassemble. Symbols, hexadecimal values, or hexadecimal expressions can be used in specification of the address.

*count* – Indicates the number of instructions to be disassembled. The value specified must be a decimal value or decimal expression.

## Aliases

**dc** – **dis**

**dpc** has no aliases.

## Example

The following is an example of how to use the **dc** and the **dpc** subcommands:

```
KDB(0)> set 4
power_pc_syntax is true
KDB(0)> dc resume_pc 10  //prints 10 instructions
.resume_pc+000000     lbz    r0,3454(0)          3454=Trconflag
.resume_pc+000004  mfsprg    r15,0
.resume_pc+000008    cmpi    cr0,r0,0
.resume_pc+00000C     lwz    toc,4208(0)         toc=TOC,4208=g_toc
.resume_pc+000010     lwz    r30,4C(r15)
.resume_pc+000014     lwz    r14,40(r15)
.resume_pc+000018     lwz    r31,8(r30)
.resume_pc+00001C     bne-   cr0.eq,<.resume_pc+0001BC>
.resume_pc+000020     lha    r28,2(r30)
.resume_pc+000024     lwz    r29,0(r14)
KDB(0)> dc mttb 5  //prints mttb function
.mttb+000000      li    r0,0
.mttb+000004   mttbl  X r0  //X shows that these instructions
.mttb+000008   mttbu  X r3  //are not supported by the current architecture
.mttb+00000C   mttbl  X r4  //POWER PC 601 processor
.mttb+000010     blr
KDB(0)> set 4  //set toggle for POWER family RS syntax
power_pc_syntax is false
KDB(0)> dc resume_pc 10  //prints 10 instructions
.resume_pc+000000     lbz    r0,3454(0)          3454=Trconflag
.resume_pc+000004   mfspr    r15,110
.resume_pc+000008    cmpi    cr0,r0,0
.resume_pc+00000C       l    toc,4208(0)         toc=TOC,4208=g_toc
.resume_pc+000010       l    r30,4C(r15)
.resume_pc+000014       l    r14,40(r15)
.resume_pc+000018       l    r31,8(r30)
.resume_pc+00001C     bne    cr0.eq,<.resume_pc+0001BC>
.resume_pc+000020     lha    r28,2(r30)
```

```
.resume_pc+000024       l    r29,0(r14)

KDB(4)> dc scdisk_pm_handler
.scdisk_pm_handler+000000     stmw    r26,FFFFFFE8(stkp)
KDB(4)> tr scdisk_pm_handler
Physical Address = 1D7CA1C0
KDB(4)> dpc 1D7CA1C0
1D7CA1C0     stmw    r26,FFFFFFE8(stkp)
```

# di subcommand

## Purpose

The **di** subcommand decodes the given hexadecimal instruction word.

## Syntax

**di** *hexadecimal_instruction*

## Parameters

- *hexadecimal_instruction* – Specifies the hexadecimal instruction word to be decoded.

The hexadecimal instruction word displays the actual instruction, with the operations code and the operands, of the given hexadecimal instruction. The **di** subcommand accepts a user input hexadecimal instruction word and decodes it into the actual instruction word in the form of the operations code and the operands.

## Aliases

**decode**

## Example

The following is an example of how to use the **di** subcommand:

```
KDB(0)> di 7Ce6212e
stwx    r7,r6,r4
KDB(0)>
```

## dr subcommand

### Purpose

The **dr** subcommand displays general purpose, segment, special, or floating point registers.

### Syntax

**dr** [**gp** | **sr** | **sp** | **fp** | **vmx** | *reg_name*]

### Parameters

- **gp** – Displays general purpose registers.
- **sr** – Displays segment registers.
- **sp** – Displays special purpose registers.
- **fp** – Displays floating point registers.
- **vmx** – Displays the current contents of vector registers. This is not the contents of the currently running thread's vector register state unless the thread is the current owner of the vector unit.
- *reg_name* – Displays a specific register by name.

The current thread context is used to locate the values to display. The **sw** subcommand can be used to change the context to other threads.

If no parameter is given, the general purpose registers are displayed.

For **BAT** registers, the **dbat** and **ibat** subcommands must be used.

### Aliases

No aliases.

### Example

1. The following is an example of how to use the **dr** subcommand:

```
KDB(0)> dr ?
Usage:       dr [sp|sr|gp|fp|hmt|vmx|<reg.name>]
Usage:       mr [sp|sr|gp|fp|<reg.name>]
   sp reg. name: iar   msr   cr    lr    ctr   xer   mq    asr
   ............. dsisr dar   dec   sdr0  sdr1  srr0  srr1  dabr
   ............. dabrx rtcu  rtcl  tbu   tbl   sprg0 sprg1 sprg2
   ............. sprg3 pir   pvr   ear   fpecr ctrl  hid0  hid1
   ............. hid4  hid5  iabr  dmiss imiss dcmp  icmp  hash1
   ............. hash2 rpa   buscsr l2cr  l2sr  imc   sia   sda
   ............. imru  imrl  mmcra mmcr0 mmcr1 pmc1  pmc2  pmc3
   ............. pmc4  pmc5  pmc6  pmc7  pmc8
   sr reg. name: s0  s1  s2  s3  s4  s5  s6  s7
   ............. s8  s9  s10 s11 s12 s13 s14 s15
   gp reg. name: r0  r1  r2  r3  r4  r5  r6  r7
   ............. r8  r9  r10 r11 r12 r13 r14 r15
   ............. r16 r17 r18 r19 r20 r21 r22 r23
   ............. r24 r25 r26 r27 r28 r29 r30 r31
   fp reg. name: f0  f1  f2  f3  f4  f5  f6  f7
   ............. f8  f9  f10 f11 f12 f13 f14 f15
   ............. f16 f17 f18 f19 f20 f21 f22 f23
   ............. f24 f25 f26 f27 f28 f29 f30 f31
   ............. fpscr
   vmx reg. name: vr0  vr1  vr2  vr3  vr4  vr5  vr6  vr7
   .............. vr8  vr9  vr10 vr11 vr12 vr13 vr14 vr15
   .............. vr16 vr17 vr18 vr19 vr20 vr21 vr22 vr23
   .............. vr24 vr25 vr26 vr27 vr28 vr29 vr30 vr31
   .............. vscr vrsave
```

```
hmt reg. name: rctrl  thctl thto  dormiar dormmsr
KDB(0)> dr //print general purpose registers
r0  : 00003730  r1  : 2FEDFF88  r2  : 00211B6C  r3  : 00000000  r4  : 00000003
r5  : 007FFFFF  r6  : 0002F930  r7  : 2FEAFFFC  r8  : 00000009  r9  : 20019CC8
r10 : 00000008  r11 : 00040B40  r12 : 0009B700  r13 : 2003FC60  r14 : DEADBEEF
r15 : 00000000  r16 : DEADBEEF  r17 : 2003FD28  r18 : 00000000  r19 : 20009168
r20 : 2003FD38  r21 : 2FEAFF3C  r22 : 00000001  r23 : 2003F700  r24 : 2FEE02E0
r25 : 2FEE0000  r26 : D0005454  r27 : 2A820846  r28 : E3000E00  r29 : E60008C0
r30 : 00353A6C  r31 : 00000511
KDB(0)> dr sp //print special registers
iar   : 10001C48  msr   : 0000F030  cr    : 28202884  lr    : 100DAF18
ctr   : 100DA1D4  xer   : 00000003  mq    : 00000DF4
dsisr : 42000000  dar   : 394A8000  dec   : 007DDC00
sdr1  : 00380007  srr0  : 10001C48  srr1  : 0000F030
dabr  : 00000000  rtcu  : 2DC05E64  rtcl  : 2E993E00
sprg0 : 000A5740  sprg1 : 00000000  sprg2 : 00000000  sprg3 : 00000000
pid   : 00000000  fpecr : 00000000  ear   : 00000000  pvr   : 00010001
hid0  : 8101FBC1  hid1  : 00004000  iabr  : 00000000
KDB(0)> dr sr //print segment registers
s0  : 60000000  s1  : 60001377  s2  : 60001BDE  s3  : 60001B7D  s4  : 6000143D
s5  : 60001F3D  s6  : 600005C9  s7  : 007FFFFF  s8  : 007FFFFF  s9  : 007FFFFF
s10 : 007FFFFF  s11 : 007FFFFF  s12 : 007FFFFF  s13 : 60000A0A  s14 : 007FFFFF
s15 : 600011D2
KDB(0)> dr fp //print floating point registers
f0  : C027C28F5C28F5C3  f1  : 000333335999999A  f2  : 3FE3333333333333
f3  : 3FC9999999999999  f4  : 7FF0000000000000  f5  : 00100000C0000000
f6  : 4000000000000000  f7  : 000000009A068000  f8  : 7FF8000000000000
f9  : 00000000BA411000  f10 : 0000000000000000  f11 : 0000000000000000
f12 : 0000000000000000  f13 : 0000000000000000  f14 : 0000000000000000
f15 : 0000000000000000  f16 : 0000000000000000  f17 : 0000000000000000
f18 : 0000000000000000  f19 : 0000000000000000  f20 : 0000000000000000
f21 : 0000000000000000  f22 : 0000000000000000  f23 : 0000000000000000
f24 : 0000000000000000  f25 : 0000000000000000  f26 : 0000000000000000
f27 : 0000000000000000  f28 : 0000000000000000  f29 : 0000000000000000
f30 : 0000000000000000  f31 : 0000000000000000  fpscr : BA411000
KDB(0)> dr ctr //print CTR register
ctr   : 100DA1D4
100DA1D4 cmpi       cr0,r3,E7           r3=2FEAB008
KDB(0)> dr msr print MSR register
msr   : 0000F030  bit set: EE PR FP ME IR DR
KDB(0)> dr cr
cr    : 28202884  bits set in CR0 : EQ
..............................CR1 : LT
..............................CR2 : EQ
..............................CR4 : EQ
..............................CR5 : LT
..............................CR6 : LT
..............................CR7 : GT
KDB(0)> dr xer //print XER register
xer   : 00000003  comparison byte: 0  length: 3
KDB(0)> dr iar //print IAR register
iar   : 10001C48
10001C48 stw        r12,4(stkp)         r12=28202884,4(stkp)=2FEAAFD4
KDB(0)> set 11 //enable 64 bits display on 620 machine
64_bit is true
KDB(0)> dr //display 620 general purpose registers
r0  : 0000000000244CF0  r1  : 0000000000259EB4  r2  : 000000000025A110
r3  : 00000000000A4B60  r4  : 0000000000000001  r5  : 0000000000000001
r6  : 00000000000000F0  r7  : 0000000000001090  r8  : 000000000018DAD0
r9  : 000000000015AB20  r10 : 000000000018D9D0  r11 : 0000000000000000
r12 : 000000000023F05C  r13 : 00000000000001C8  r14 : 00000000000000BC
r15 : 0000000000000040  r16 : 0000000000000040  r17 : 00000000080300F0
r18 : 0000000000000000  r19 : 0000000000000000  r20 : 0000000000225A48
r21 : 0000000001FF3E00  r22 : 00000000002259D0  r23 : 000000000025A12C
r24 : 0000000000000001  r25 : 0000000000000001  r26 : 0000000001FF42E0
r27 : 0000000000000000  r28 : 0000000001FF4A64  r29 : 0000000001FF4000
r30 : 00000000000034CC  r31 : 0000000001FF4A64
```

```
KDB(0)> dr sp display 620 special registers
iar   : 000000000023F288  msr   : 0000000000021080  cr    : 42000440
lr    : 0000000000245738  ctr   : 0000000000000000  xer   : 00000000
mq    : 00000000  asr   : 0000000000000000
dsisr : 42000000  dar   : 00000000000000EC  dec   : C3528E2F
sdr1  : 01EC0000  srr0  : 000000000023F288  srr1  : 0000000000021080
dabr  : 0000000000000000  tbu   : 00000002  tbl   : AF33287B
sprg0 : 00000000000A4C00  sprg1 : 0000000000000040
sprg2 : 0000000000000000  sprg3 : 0000000000000000
pir   : 0000000000000000  ear   : 00000000  pvr   : 00140201
hid0  : 7001C080  iabr  : 0000000000000000
buscsr : 00000000008DC800  l2cr  : 000000000000421A  l2sr  : 0000000000000000
mmcr0 : 00000000  pmc1  : 00000000  pmc2  : 00000000
sia   : 0000000000000000  sda   : 0000000000000000
KDB(0)>
```

2. The following is an example of how to use the **dr** subcommand on a PCI machine to print one word at physical address 80000cfc::

```
KDB(0)> ddpw 80000cfc //Print one word at physical address 80000cfc
80000CFC: D0000080   //Read is done in relocated mode, cache inhibited
KDB(0)>
```

## ddvb, ddvh, ddvw, ddvd, ddpb, ddph, ddpw, and ddpd subcommand

### Purpose

The **ddvb**, **ddvh**, **ddvw** and **ddvd** subcommands can be used to access memory in translated mode, using an effective address already mapped. On a 64-bit machine, double words correctly aligned are accessed in a single load (ld) instruction with the **ddvd** subcommand.

The **ddpb**, **ddph**, **ddpw** and **ddpd** subcommands can be used to access memory in translated mode, using a physical address that will be mapped. On a 64-bit machine, double words correctly aligned are accessed in a single load (ld) instruction with the **ddpd** subcommand. The DBAT interface is used to translate this address in cache-inhibited mode.

**Note:** These subcommands are only available within the KDB kernel debugger. They are not included in the **kdb** command.

### Syntax

**ddvb** *EffectiveAddress* [*count*]

**ddvh** *EffectiveAddress* [*count*]

**ddvw** *EffectiveAddress* [*count*]

**ddvd** *EffectiveAddress* [*count*]

**ddpb** *PhysicalAddress* [*count*]

**ddph** *PhysicalAddress* [*count*]

**ddpw** *PhysicalAddress* [*count*]

**ddpd** *PhysicalAddress* [*count*]

### Parameters

- *EffectiveAddress* – Specifies the effective or virtual address of the starting memory area to display. Symbols, hexadecimal values, or hexadecimal expressions can be used to specify the address.
- *PhysicalAddress* – Specifies the physical or real address of the starting memory area to display. Symbols, hexadecimal values, or hexadecimal expressions can be used to specify the address.
- *count* – Specifies the number of bytes for **ddvb** and **ddpb** to display, specifies the number of half words for **ddvh** and **ddph** to display, specifies the number of words for **ddvw** and **ddpw** to display and specifies the number of double words for **ddvd** and **ddpd** to display. The *count* argument is a hexadecimal value.

I/O space memory (Direct Store Segment (T=1)) cannot be accessed when translation is disabled. The areas mapped by the **bat** command areas must also be accessed with translation enabled. Otherwise, cache controls are ignored.

**Note:** The subcommands that use effective addresses assume that mapping to real addresses is currently valid. No check is done by the KDB kernel debugger. The subcommands that use real addresses can be used to let KDB kernel debugger perform the mapping (attach and detach).

## Aliases

The alias for:

- **ddvb** is **diob**
- **ddvh** is **dioh**
- **ddvw** is **diow**
- **ddvd** is **diod**

There are no aliases for the following:

- **ddpb**
- **ddph**
- **ddpw**
- **ddpd**

## Example

**Note:** The PowerPC 601 RISC Microprocessor is only available on AIX 5.1 and earlier.

The following is an example on the PowerPC 601 RISC Microprocessor:

```
KDB(0)> tr fff19610 //show current mapping
BAT mapping for FFF19610
DBAT0 FFC0003A FFC0005F
 bepi 7FE0 brpn 7FE0 bl 001F v 1 wim 3 ks 1 kp 0 pp 2 s 0
 eaddr = FFC00000, paddr = FFC00000 size = 4096 KBytes
KDB(0)> ddvb fff19610 10 //print 10 bytes using data relocate mode enable
FFF19610: 0041 96B0  6666 CEEA  0041 A0B0  0041 AAB0      .A..ff...A...A..
KDB(0)> ddvw fff19610 4 //print 4 words using data relocate mode enable
FFF19610: 004196B0 76763346 0041A0B0 0041AAB0
KDB(0)>
```

The following is an example on a PCI machine:

```
KDB(0)> ddpw 80000cfc //print one word at physical address 80000cfc
80000CFC: D0000080    //Read is done in relocated mode, cache inhibited
KDB(0)>
```

# Chapter 15. Memory search and extract subcommands

The subcommands in this category are used to search and extract information from memory. These subcommands include the following:

- find
- findp
- ext
- extp

 

# find and findp subcommands

## Purpose

The **find** and **findp** subcommands search for a specific pattern in memory.

## Syntax

**find** [**-s** *string*]

**find** *effectiveaddress pattern* [*mask* | *delta*]

**findp** [**-s***string*]

**findp***physicaladdress pattern* [*mask* | *delta*]

## Parameters

- **-s** – Indicates the pattern to be searched for is an ASCII string.
- *EffectiveAddress* – Specifies the effective or virtual address. Symbols, hexadecimal values, or hexadecimal expressions can be used to specify the address.
- *PhysicalAddress* – Specifies the physical or real address. Symbols, hexadecimal values, or hexadecimal expressions can be used to specify the address.
- *string* – Specifies the ASCII string to search for if the **-s** option is specified. The period (.) is used to match any character.
- *pattern* – Specifies the hexadecimal value of the pattern to search for. The pattern is limited to one word in length.
- *mask* – If a pattern is specified, a mask can be specified to eliminate bits from consideration for matching purposes. This parameter is a one-word hexadecimal value.
- *delta* – Specifies the increment to move forward after an unsuccessful match. This parameter is a one-word hexadecimal value.

The pattern that is searched for can either be an ASCII string, if the **-s** option is used, or a one word hexadecimal value. If the search is for an ASCII string, the period (.) can be used to match any character.

A mask parameter can be used if the search is for a hexadecimal value. The mask is used to eliminate bits from consideration. When it is checking for matches, the value from memory is ended with the mask and then compared to the specified pattern for matching. For example, a mask of `7fffffff` indicates that the high bit is not to be considered. If the specified pattern was `0000000d` and the mask was `7fffffff`, the values `0000000d` and `8000000d` are both considered matches.

A parameter can also be specified to indicate the delta that is applied to determine the next address to check for a match. This ensures that the matching pattern occurs on specific boundaries. For example, if you want to find the `0f0000ff` pattern aligned on a 64-byte boundary, the following subcommand could be used:

```
find 0f0000ff ffffffff 40
```

The default delta is one byte for matching strings and one word for matching a specified hexadecimal pattern.

If the **find** or **findp** subcommands find the specified pattern, the data and address are displayed. Continue the search from that point by pressing the Enter key.

## Aliases

No aliases.

## Example

The following is an example of how to use the **find** and the **findp** subcommands:

```
KDB(0)> tpid  //print current thread
              SLOT NAME     STATE   TID PRI CPUID CPU FLAGS    WCHAN

thread+002F40   63*nfsd     RUN   03F8F 03C       000 00000000
KDB(0)> find lock_pinned 03F8F 00ffffff 20  //search TID in the lock area
   //compare only 24 low bits, on cache aligned addresses (delta 0x20)
lock_pinned+00D760: 00003F8F 00000000 00000005 00000000
KDB(0)>  <CR/LF>  //repeat last command
Invalid address E800F000, skip to (^C to interrupt)
............... E8800000
Invalid address E8840000, skip to (^C to interrupt)
............... E9000000
Invalid address E9012000, skip to (^C to interrupt)
............... F0000000
KDB(0)> findp 0 E819D200  //search in physical memory
00F97C7C: E819D200 00000000 00000000 00000000
KDB(0)> <CR/LF>  //repeat last command
05C4FB18: E819D200 00000000 00000000 00000000
KDB(0)> <CR/LF>  //repeat last command
0F7550F0: E819D200 00000000 E60009C0 00000000
KDB(0)> <CR/LF>  //repeat last command
0F927EE8: E819D200 00000000 05E62D28 00000000
KDB(0)> <CR/LF>  //repeat last command
0FAE16E8: E819D200 00000000 05D3B528 00000000
KDB(0)> <CR/LF>  //repeat last command
kdb_get_real_memory: Out of range address 1FFFFFFF
KDB(0)>
KDB(0)>find -s 01A86260 pse  //search "pse" in pse text code
01A86ED4: 7073 655F  6B64 6200  8062 0518  8063 0000   pse_kdb..b...c..
KDB(0)>  <CR/LF>  //repeat last command
01A92952: 7073 6562  7566 6361  6C6C 735F  696E 6974   psebufcalls_init
KDB(0)>  <CR/LF>  //repeat last command
01A939AE: 7073 655F  6275 6663  616C 6C00  0000 BF81   pse_bufcall.....
KDB(0)>  <CR/LF>  //repeat last command
01A94F5A: 7073 655F  7265 766F  6B65 BEA1  FFD4 7D80   pse_revoke....}.
KDB(0)>  <CR/LF>  //repeat last command
01A9547E: 7073 655F  7365 6C65  6374 BE41  FFC8 7D80   pse_select.A..}.
KDB(0)> find -s 01A86260 pse_....._thread  //how to use '.'
01A9F586: 7073 655F  626C 6F63  6B5F 7468  7265 6164   pse_block_thread
KDB(0)>  <CR/LF>  //repeat last command
01A9F6EA: 7073 655F  736C 6565  705F 7468  7265 6164   pse_sleep_thread
```

# ext and extp subcommands

## Purpose

The **ext** and **extp** subcommands display a specific area from a structure.

## Syntax

**ext** [**-p**] *EffectiveAddress delta* [*size* | *count*]

**extp** [**-p**] *PhysicalAddress delta* [*size* | *count*]

## Parameters

- **-p** – Indicates that the delta argument is the offset to a pointer to the next area.
- *EffectiveAddress* – Specifies the effective or virtual address at which to begin displaying values. Symbols, hexadecimal values, or hexadecimal expressions can be used to specify the address.
- *PhysicalAddress* – Specifies the physical or real address at which to begin displaying values. Symbols, hexadecimal values, or hexadecimal expressions can be used to specify the address.
- *delta* – Specifies the offset to the next area to be displayed or the offset from the beginning of the current area to a pointer to the next area. This argument is a hexadecimal value.
- *size* – Specifies the hexadecimal value that indicates the number of words to display.
- *count* – Specifies the hexadecimal value that indicates the number of entries to traverse.

If the **-p** flag is not specified, these subcommands display the number of words indicated in the size argument. They then increment the address by the delta and display the data at that location. This procedure is repeated for the number of times indicated in the *count* parameter.

If the **-p** flag is specified, these subcommands display the number of words indicated by the *size* parameter. The next address from which data is to be displayed is then determined by using the value at the current address plus the offset indicated in the *delta* parameter (for example, `*(addr+delta)`). This procedure is repeated for the number of times indicated in the *count* parameter.

If an array exists, it can be traversed displaying the specified area for each entry of the array. These subcommands can also be used to traverse a linked list displaying the specified area for each entry.

## Aliases

No aliases.

## Example

The following is an example of how to use the **exp** and the **expt** subcommands:

```
KDB(0)> ppda

Per Processor Data Area [0101A9C0]
csa.............000000000184EE00
mstack..........000000000184BE00
fpowner.........0000000000000000
curthread.......F100060004066400
syscall.........00000000003CDA21
worst_run_pri................00FF
run_pri.......................FF
v_pnda..........000000000126CCB0
cpunidx......................0000
wait_thread.....F100060004066400
ppda_pal[0]..............00000000
ppda_pal[1]..............00000000
ppda_pal[2]..............00000000
```

```
ppda_pal[3]..............00000000
phy_cpuid....................0000
sradid.......................0000
pvpa.............0000000001130400
slb_reload...................0000
slb_index....................0000
slb_stoimask.................0000
slb_stoibits.................0000
slb_stab_mask....0000000000000000
slb_g_start......0000000000000000
slb_g_nesids.....0000000000000000
slb_ksp_start....0000000000000000
slb_ksp_nesids...0000000000000000
slb_glp_start....0000000000000000
slb_glp_nesids...0000000000000000
slb_glp_tbl......0000000000000000
slb_lgpg_start...0000000000000000
slb_lgpg_nesids..0000000000000000
slb_slbsave......0000000000000000
slb_recurse_cnt..............0000
slb_stab_addr....0000000000000000
KDB(0)> ext -p 000000000184EE00 0 10 2 // csa address from the ppda
mststack+020E00: F0000000 2FF47600 00000000 00000000 ..../.v.........
mststack+020E10: 00000000 00000000 00000000 00000000 ................
mststack+020E20: 00000000 00000000 A0000000 000010B2 ................
mststack+020E30: 00000000 000302A0 00000000 0003023C ...............<

__ublock+000000: 00000000 00000000 00000000 00000000 ................
__ublock+000010: 00000000 00000000 00000000 00000000 ................
__ublock+000020: 0B000000 00000000 A0000000 00009032 ...............2
__ublock+000030: 00000000 00025138 00000000 00028828 ......Q8.......(

  KDB(0)> ext 000000000184BE00 3000 10 2 // mstsave address from the ppda
mststack+01DE00: 00000000 0184EE00 00000000 00000000 ................
mststack+01DE10: 00000000 00000000 00000000 00000000 ................
mststack+01DE20: 00000000 00000000 A0000000 000090B2 ................
mststack+01DE30: 00000000 0000944C 00000000 0009A798 .......L........
mststack+020E00: F0000000 2FF47600 00000000 00000000 ..../.v.........
mststack+020E10: 00000000 00000000 00000000 00000000 ................
mststack+020E20: 00000000 00000000 A0000000 000010B2 ................
mststack+020E30: 00000000 000302A0 00000000 0003023C ...............<

KDB(0)>
```

# Chapter 16. Memory modification subcommands

The subcommands in this category are used to modify memory. These subcommands include the following:

- m
- mw
- md
- mpw
- mpd
- st
- stc
- sth
- mdvb
- mdvh
- mdvw
- mdvd
- mdpb
- mdph
- mdpw
- mdpd
- mr

 

## m, mw, md, mp, mpw, and mpd subcommands

### Purpose

The **m** (modify bytes), **mw** (modify words) and **md** (modify double words) subcommands modify memory starting at a specified effective address. The **mp** (modify bytes), **mpw** (modify words) and **mpd** (modify double words) subcommands modify memory starting at a specified real address.

These subcommands are only available within the KDB kernel debugger. They are not included in the **kdb** command.

### Syntax

**m** *effectiveaddress*

**mw** *effectiveaddress*

**md** *effectiveaddress*

**mp** *physicaladdress*

**mpw** *physicaladdress*

**mpd** *physicaladdress*

### Parameters

- *effectiveaddress* – Specifies the effective or virtual address of the starting memory area to modify. Symbols, hexadecimal values, or hexadecimal expressions can be used to specify the address.
- *physicaladdress* – Specifies the physical or real address of the starting memory area to modify. Symbols, hexadecimal values, or hexadecimal expressions can be used to specify the address.

Read or write access can be in virtual or real mode.

These subcommands are interactive. Each modification is entered one-by-one. The first unexpected input stops modification. For example, a period (.) can be used to indicate the end of the data. If a break point is set at the same address, use the **mw** subcommand to maintain break point coherency.

**Note:** Symbolic expressions are not allowed as input.

### Aliases

No aliases.

### Example

The following is an example of how to use the **mw** and **m**subcommands to do a patch:

```
KDB(0)> dc @iar //print current instruction
   .open+000000    mflr    r0
   KDB(0)> mw @iar //nop current instruction
   .open+000000:  7C0802A6  = 60000000
   .open+000004:  93E1FFFC  = . //end of input
   KDB(0)> dc @iar //print current instruction
   .open+000000     ori    r0,r0,0
   KDB(0)> m @iar //restore current instruction byte per byte
   .open+000000:  60  = 7C
   .open+000001:  00  = 08
   .open+000002:  00  = 02
   .open+000003:  00  = A6
```

```
.open+000004:  93  = . //end of input
KDB(0)> dc @iar //print current instruction
.open+000000    mflr    r0
KDB(0)> tr @iar //physical address of current instruction
Physical Address = 001C5BA0
KDB(0)> mwp 001C5BA0 //modify with physical address
001C5BA0:  7C0802A6  = <CR/LF>
001C5BA4:  93E1FFFC  = <CR/LF>
001C5BA8:  90010008  = <CR/LF>
001C5BAC:  9421FF40  = 60000000
001C5BB0:  83E211C4  = . //end of input
KDB(0)> dc @iar 5 //print instructions
.open+000000    mflr    r0
.open+000004     stw    r31,FFFFFFFC(stkp)
.open+000008     stw    r0,8(stkp)
.open+00000C     ori    r0,r0,0
.open+000010     lwz    r31,11C4(toc)       11C4(toc)=_open$$
KDB(0)> mw open+c //restore instruction
.open+00000C:  60000000  = 9421FF40
.open+000010:  83E211C4  = . //end of input
KDB(0)> dc open+c //print instruction
.open+00000C    stwu    stkp,FFFFFF40(stkp)
KDB(0)>
```

# st, stc, and sth subcommands

## Purpose

The **st**, **stc** and **sth** subcommands store data at a specified address.

## Syntax

**st** *EffectiveAddress Value*

**stc** *EffectiveAddress Value*

**sth** *EffectiveAddress Value*

## Parameters

- *EffectiveAddress* – Specifies the effective address to which the data will be stored. Hexadecimal values or hexadecimal expressions can be used in specification of the address.
- *Value* – Specifies the data value to be stored. The value stored is:
  - One word if you use the **st** subcommand
  - One character if you use the **stc** subcommand
  - One half-word if you use the **sth** subcommand

## Aliases

No aliases.

## Example

The following is an example of how to use the **st**, the **stc** and the **sth** subcommands:

```
KDB(0)> dw 20
 00000020: 00000000 00000000 00000000 00000000  ................
 KDB(0)> st 20 11111111
 KDB(0)> dw 20
 00000020: 11111111 00000000 00000000 00000000  ................
 KDB(0)> st 20 2
 KDB(0)> dw 20
 00000020: 00000002 00000000 00000000 00000000  ................
 KDB(0)> st 20 0
 KDB(0)> dw 20
 00000020: 00000000 00000000 00000000 00000000  ................
 KDB(0)> stc 20 33
 KDB(0)> dw 20
 00000020: 33000000 00000000 00000000 00000000  3...............
 KDB(0)> st 20 0
 KDB(0)> dw 20
 00000020: 00000000 00000000 00000000 00000000  ................
 KDB(0)> sth 20 4444
 KDB(0)> dw 20
 00000020: 44440000 00000000 00000000 00000000  DD..............
 KDB(0)> st 20 0
 KDB(0)> dw 20
 00000020: 00000000 00000000 00000000 00000000  ................
```

## mdvb, mdvh, mdvw, mdvd, mdpb, mdph, mdpw, mdpd subcommands

### Purpose

The **mdvb**, **mdvh**, **mdvw**, and **mdvd** subcommands can be used to access memory in translated mode, using an effective address already mapped. On a 64-bit machine, double words are accessed by the **mdvd** subcommand in a single store instruction.

The **mdpb**, **mdph**, **mdpw**, and **mdpd** subcommands access memory in translated mode, using a physical address that will be mapped. On a 64-bit machine, correctly-aligned double words are accessed by the **mdpd** subcommand in a single store instruction. The DBAT interface is used to translate this address in cache-inhibited mode.

**Note:** These subcommands are only available within the KDB kernel debugger. They are not included in the kdb command.

### Syntax

**mdvb** *effectiveaddress*

**mdvh** *effectiveaddress*

**mdvw** *effectiveaddress*

**mdvd** *effectiveaddress*

**mdpb** *physicaladdress*

**mdph** *physicaladdress*

**mdpw** *physicaladdress*

**mdpd** *physicaladdress*

### Parameters

- *effectiveaddress* – Specifies the virtual (effective) address of the memory to modify. It can be symbols, hexadecimal values, or hexadecimal expressions.
- *physicaladdress* – Specifies the real (physical) address of the memory to modify. It can be symbols, hexadecimal values, or hexadecimal expressions.

These subcommands are available to write in I/O space memory.

To avoid bad effects, memory is not read before, only the specified write is performed with translation enabled. Access can be in bytes, half words, words or double words.

**Note:** The subcommands using effective addresses assume that mapping to real addresses is currently valid. No check is done by KDB kernel debugger. The subcommands using real addresses allow KDB kernel debugger to do the mapping (attach and detach).

### Aliases

The aliases are:
**mdvb** – **miob**
**mdvh** – **mioh**
**mdvw** – **miow**
**mdvd** – **miod**

There are no aliases for the following:

**mdpb**
**mdph**
**mdpw**
**mdpd**

## Example

The following is an example on the PowerPC 601 RISC Microprocessor:

**Note:** The PowerPC 601 RISC Microprocessor is only supported on AIX 5.1 and earlier.

```
KDB(0)> tr FFF19610 //print physical mapping
   BAT mapping for FFF19610
   DBAT0 FFC0003A FFC0005F
    bepi 7FE0 brpn 7FE0 bl 001F v 1 wim 3 ks 1 kp 0 pp 2 s 0
    eaddr = FFC00000, paddr = FFC00000 size = 4096 KBytes
   KDB(0)> mdvb fff19610 //byte modify with data relocate enable
   FFF19610: ?? = 00
   FFF19611: ?? = 00
   FFF19612: ?? = . //end of input
   KDB(0)> mdvw fff19610 //word modify with data relocate enable
   FFF19610: ???????? = 004196B0
   FFF19614: ???????? = . //end of input
   KDB(0)>
```

The following is an example on a PCI machine:

```
KDB(0)> mdpw 80000cf8 //change one word at physical address 80000cf8
   80000CF8: ???????? = 84000080
   80000CFC: ???????? = . //Write is done in relocated mode, cache inhibited
   KDB(0)> ddpw 80000cfc //print one word at physical address 80000cfc
   80000CFC: D2000000
   KDB(0)> mdpw 80000cfc //change one word at physical address 80000cfc
   80000CFC: ???????? = d0000000
   80000D00: ???????? = .
   KDB(0)> mdpw 80000cf8 //change one word at physical address 80000cf8
   80000CF8: ???????? = 8c000080
   80000CFC: ???????? = .
   KDB(0)> ddpw 80000cfc //print one word at physical address 80000cfc
   80000CFC: D2000080
```

# mr subcommand

## Purpose

The **mr** subcommand modifies general purpose, segment, special, or floating point registers.

## Syntax

**mr** [**gp** | **sr** | **sp** | **fp** | *reg_name*]

## Parameters

- **gp** – Modifies general purpose registers.
- **sr** – Modifies segment registers.
- **sp** – Modifies special purpose registers.
- **fp** – Modifies floating point registers.
- *reg_name* – Modifies a specific register by name.

Individual registers can also be selected for modification by register name. The current thread context is used to locate the register values to be modified. Use the **sw** subcommand to change the context to other threads. When the register being modified is in the **mst** subcommand context, the KDB kernel debugger alters the Machine Save State Area. When the register being modified is a special register, the register is altered immediately. Symbolic expressions are allowed as input.

If the **gp**, **sr**, **sp**, or **fp** options are used, modification of all of the registers in the group is allowed. The current value for a single register is shown and modification is allowed. Then, the value for the next register is displayed for modification. Entry of an invalid character, such as a period (.), ends modification of the registers. If the value for a register is to be left unmodified, press Enter to continue to the next register for modification.

## Aliases

No aliases.

## Example

The following is an example of how to use the **mr** subcommand:

```
KDB(0)> dc @iar  //print current instruction
.open+000000    mflr    r0
KDB(0)> mr iar  //modify current instruction address
iar : 001C5BA0 = @iar+4
KDB(0)> dc @iar  //print current instruction
.open+000004     stw    r31,FFFFFFFC(stkp)
KDB(0)> mr iar  //restore current instruction address
iar : 001C5BA4 = @iar-4
KDB(0)> dc @iar  //print current instruction
.open+000000    mflr    r0
KDB(0)> mr sr  //modify first invalid segment register
s0  : 00000000 = <CR/LF>
s1  : 60000323 = <CR/LF>
s2  : 20001E1E = <CR/LF>
s3  : 007FFFFF = 0
s4  : 007FFFFF = .  //end of input
KDB(0)> dr s3  //print segment register 3
s3  : 00000000
KDB(0)> mr s3  //restore segment register 3
s3  : 00000000 = 007FFFFF
KDB(0)> mr f29  //modify floating point register f29
f29 : 0000000000000000 = 000333335999999A
KDB(1)> mr vr0 //modify vector register vr0
```

```
vr0 : 00000000000000000000000000000000 = 1122334455667788 <CR/LF>
 = 99aabbccddeeff00
KDB(0)> dr f29
f29 : 000333335999999A
KDB(1)> dr vr0 //dump vector register vr0
vr0 : 112233445566778899AABBCCDDEEFF00
KDB(0)> u
Uthread [2FF3B400]:
   save@......2FF3B400   fpr@.......2FF3B550
...
KDB(0)> dd 2FF3B550 20
__ublock+000150: C027C28F5C28F5C3 000333335999999A  .'..\(....33Y...
__ublock+000160: 3FE3333333333333 3FC9999999999999  ?.333333?.......
__ublock+000170: 7FF0000000000000 00100000C0000000  ................
__ublock+000180: 4000000000000000 000000009A068000  @...............
__ublock+000190: 7FF8000000000000 00000000BA411000  .............A..
__ublock+0001A0: 0000000000000000 0000000000000000  ................
__ublock+0001B0: 0000000000000000 0000000000000000  ................
__ublock+0001C0: 0000000000000000 0000000000000000  ................
__ublock+0001D0: 0000000000000000 0000000000000000  ................
__ublock+0001E0: 0000000000000000 0000000000000000  ................
__ublock+0001F0: 0000000000000000 0000000000000000  ................
__ublock+000200: 0000000000000000 0000000000000000  ................
__ublock+000210: 0000000000000000 0000000000000000  ................
__ublock+000220: 0000000000000000 0000000000000000  ................
__ublock+000230: 0000000000000000 000333335999999A  ..........33Y...
__ublock+000240: 0000000000000000 0000000000000000  ................
KDB(0)>
```

**Note:** The **vr0** register modifies the current vector register contents. The vector register state of the current thread is not modified unless the thread is the current owner of the vector unit. The 16-byte vector input is entered as 8 bytes followed by a carriage return and then followed by 8 bytes.

# Chapter 17. Breakpoint and steps subcommands

The subcommands in this category are used to set and clear breakpoints and provide step functions. These subcommands include the following:

- b
- lb
- c
- lc
- ca
- r
- gt
- n
- s
- S
- B

 

# b subcommand

## Purpose

The **b** subcommand sets a permanent global breakpoint in the code. KDB kernel debugger checks whether a valid instruction is trapped.

**Note:** This subcommand is only available within the KDB kernel debugger. It is not included in the **kdb** command.

## Syntax

**b** [**-p** | **-v**] [ *address*]

## Parameters

- **-p** – Indicates that the breakpoint address is a physical or real address.
- **-v** – Indicates that the breakpoint address is a effective or virtual address.
- *address* – Specifies the address of the breakpoint. This may either be a physical address or a virtual address. Symbols, hexadecimal values, or hexadecimal expressions can be used to specify the address.

If an invalid instruction is detected, a warning message is displayed. If the warning message is displayed, the breakpoint should be removed; otherwise, memory can be corrupted.

## Aliases

**brk**

## Example

The following example is before VMM setup:

```
KDB(0)> b vsi  //set break point on vsi()
.vsi+000000 (real address:002AA5A4) permanent & global
KDB(0)> e  //exit debugger
...
Breakpoint
.vsi+000000     stmw    r29,FFFFFFF4(stkp)  <.mainstk+001EFC> r29=isync_sc1+000040,FFFFFFF4(stkp)=.mainstk+001EFC
```

The following example is after VMM setup:

```
KDB(0)> b  //display current active break points
No breakpoints are set.
KDB(0)> b 0  //set break point at address 0
WARNING: break point at 00000000 on invalid instruction (00000000)
00000000 (sid:00000000) permanent & global
KDB(0)> c 0  //remove break point at address 0
KDB(0)> b vmvcs  //set break point on vmvcs()
.vmvcs+000000 (sid:00000000) permanent & global
KDB(0)> b i_disable  //set break point on i_disable()
.i_disable+000000 (sid:00000000) permanent & global
KDB(0)> e  //exit debugger
...
Breakpoint
.i_disable+000000   mfmsr    r7                 <start+001008> r7=DEADBEEF
KDB(0)> b  //display current active break points
0:      .vmvcs+000000 (sid:00000000)  permanent & global
1:      .i_disable+000000 (sid:00000000)  permanent & global
KDB(0)> c 1  //remove break point slot 1
KDB(0)> b  //display current active break points
0:      .vmvcs+000000 (sid:00000000)  permanent & global
KDB(0)> e  //exit debugger
```

```
...
Breakpoint
.vmvcs+000000    mflr    r10                 <.initcom+000120>
KDB(0)> ca  //remove all break points
```

## lb subcommand

### Purpose

The **lb** subcommand sets a permanent local breakpoint in the code for a specific context.

**Note:** This subcommand is only available within the KDB kernel debugger. It is not included in the **kdb** command.

### Syntax

**lb** [**-p** | **-v**] [*address*]

### Parameters

- **-p** – Indicates that the breakpoint address is a physical or real address.
- **-v** – Indicates that the breakpoint address is an effective or virtual address.
- *address* – Specifies the address of the breakpoint. This can be either an effective or physical address. Symbols, hexadecimal values, or hexadecimal expressions can be used to specify the address.

The context can either be CPU-based or thread-based. Either context is controllable through a "set subcommand" on page 44 option. Each **lb** subcommand associates one context with the local breakpoint and up to eight different contexts can be set for each local breakpoint. The context is the effective address of the current thread entry in the thread table or the current processor number.

If the **lb** subcommand is used with no parameters, all current trace and breakpoints are displayed.

If an address is specified, the break is set with the context of the current thread or CPU. To set a break using a context other than the current thread or CPU, change the current context using the "sw subcommand" on page 60 and the "cpu subcommand" on page 63.

If a local breakpoint is hit with a context that was not specified, a message is displayed, but a break does not occur.

By default, KDB kernel debugger chooses the current state of the machine. If the subcommand is entered before VMM initialization, the address is the physical address or real address. If the subcommand is entered after VMM initialization, the address is the effective or virtual address. After VMM is setup, the **-p** parameter must be used to set a breakpoint in real-mode for code that is not mapped V=R. Otherwise, the KDB kernel debugger expects a virtual address and translates the address.

### Aliases

**lbrk**

### Example

The following is an example of how to use the **lb** subcommand:

```
KDB(0)> b execv  //set break point on execv()
Assumed to be [External data]: 001F4200 execve
Ambiguous: [Ext func]
001F4200 .execve
.execve+000000 (sid:00000000) permanent & global
KDB(0)> e  //exit debugger
...
Breakpoint
.execve+000000    mflr    r0                 <.svc_flih+00011C>
KDB(0)> ppda  //print current processor data area

Per Processor Data Area [00086E40]
```

```
csa......................2FEE0000  mstack...................0037CDB0
fpowner..................00000000  curthread................E60008C0
...
KDB(0)> lb kexit  //set local break point on kexit()
.kexit+000000 (sid:00000000) permanent & local < ctx: thread+0008C0 >
KDB(0)> b  //display current active break points
0:      .execve+000000 (sid:00000000)  permanent & global
1:      .kexit+000000 (sid:00000000)  permanent & local < ctx: thread+0008C0 >
KDB(0)> e  //exit debugger
...
Warning, breakpoint ignored (context mismatched):
.kexit+000000    mflr    r0                 <._exit+000020>
Breakpoint
.kexit+000000    mflr    r0                 <._exit+000020>
KDB(0)> ppda  //print current processor data area

Per Processor Data Area [00086E40]

csa......................2FEE0000  mstack...................0037CDB0
fpowner..................00000000  curthread................E60008C0
...
KDB(0)> lc 1 thread+0008C0  //remove local break point slot 1
```

# c, lc, and ca subcommands

## Purpose

The **c**, **lc** and **ca** subcommands clear breakpoints.

**Note:** This subcommand is only available within the KDB kernel debugger. It is not included in the **kdb** command.

## Syntax

**c** [*slot* | [**-p** | **-v**] *Address*]

**ca**

**lc** [*slot* | [**-p** | **-v**] *Address* [*ctx*]]

## Parameters

- **-p** – Indicates that the breakpoint address is a physical or real address.
- **-v** – Indicates that the breakpoint address is an effective or virtual address.
- *slot* – Specifies the slot number of the breakpoint. This parameter must be a decimal value.
- *Address* – Specifies the address of the breakpoint. This may either be a physical or virtual address. Symbols, hexadecimal values, or hexadecimal expressions can be used to specify the address.
- *ctx* – Specifies the context to be cleared for a local break. The context may either be a CPU or thread specification.

The **ca** subcommand erases all breakpoints. The **c** and **lc** subcommands erase only the specified breakpoint. The **c** subcommand clears all contexts for a specified breakpoint. The **lc** subcommand can be used to clear a single context for a breakpoint. If a specific context is not specified, the current context is used to determine which local breakpoint context to remove.

By default, the KDB kernel debugger chooses the current state of the machine. If the subcommand is entered before VMM initialization, the address is the physical or real address. If the subcommand is entered after VMM initialization, the address is the effective or virtual address.

**Note:** Slot numbers are not fixed. To clear slot `1` and slot `2` type `c 2; c 1` or type `c 1; c 1`. Do not enter `c 1; c 2`.

## Aliases

**c** – **cl**

**lc** – **lcl**

## Example

The following is an example of how to use the **c** and the **ca** subcommands:

```
KDB(1)> b  //list breakpoints
0:      .halt_display+000000 (sid:00000000) permanent & global
1:      .v_exception+000000 (sid:00000000) permanent & global
2:      .v_loghalt+000000 (sid:00000000) permanent & global
3:      .p_slih+000000 (sid:00000000)  trace {hit: 0}
KDB(1)> c 2  //clear breakpoint slot 2
0:      .halt_display+000000 (sid:00000000) permanent & global
1:      .v_exception+000000 (sid:00000000) permanent & global
2:      .p_slih+000000 (sid:00000000)  trace {hit: 0}
KDB(1)> c v_exception  //clear breakpoint set on v_exception
```

```
0:      .halt_display+000000 (sid:00000000) permanent & global
1:      .p_slih+000000 (sid:00000000)  trace {hit: 0}
KDB(1)> ca  //clear all breakpoints
0:      .p_slih+000000 (sid:00000000)  trace {hit: 0}
```

# r and gt subcommands

## Purpose

The **r** and **gt** subcommands set non-permanent breakpoints. Non-permanent breakpoints are local breakpoints that are cleared after they are used.

**Note:** This subcommand is only available within the KDB kernel debugger. It is not included in the **kdb** command.

## Syntax

**r**

**gt** [**-p** | **-v**] [*address*]

## Parameters

- **-p** – Indicates that the breakpoint address is a physical or real address.
- **-v** – Indicates that the breakpoint address is an effective or virtual address.
- *address* – Specifies the address of the breakpoint. This may either be a physical or real address. Symbols, hexadecimal values, or hexadecimal expressions may be used in specification of the address.

The **r** subcommand sets a breakpoint on the address found in the **lr** register. In the SMP environment, it is possible to reach this breakpoint on another processor. For this reason, it is important to use the thread or process local break point.

The **gt** subcommand performs the same function as the **r** subcommand, but the *address* must be specified for the **gt** subcommand.

By default, the KDB kernel debugger chooses the current state of the machine. If the subcommand is entered before VMM initialization, the address is physical. If the subcommand is entered after VMM initialization, the address is virtual (effective address). After VMM is initialized, the **-p** flag must be used to set a breakpoint in real-mode code that is not mapped V=R, otherwise KDB kernel debugger expects a virtual address and translates the address.

## Aliases

**r** – **return**

**gt** has no aliases.

## Example

The following is an example of how to use the **r** and the **gt** subcommands:

```
KDB(2)> b _iput  //enable break point on _iput()
._iput+000000 (sid:00000000) permanent & global
KDB(2)> e  //exit debugger
...
Breakpoint
._iput+000000     stmw    r29,FFFFFFF4(stkp)  <2FF3B1CC> r29=0A4C6C20,FFFFFFF4(stkp)=2FF3B1CC
KDB(6)> f
thread+014580 STACK:
[0021632C]_iput+000000 (0A4C6C20, 0571A808 [??])
[00263EF4]jfs_rele+0000B4 (??)
[00220B58]vnop_rele+000018 (??)
[00232178]vno_close+000058 (??)
[002266C8]closef+0000C8 (??)
[0020C548]closefd+0000BC (??, ??)
```

```
[0020C70C]close+000174 (??)
[000037C4].sys_call+000000 ()
[D000715C]fclose+00006C (??)
[10000580]10000580+000000 ()
[10000174]__start+00004C ()
KDB(6)> r  //go to the end of the function
...
.jfs_rele+0000B8        b    <.jfs_rele+00007C>  r3=0
KDB(7)> e  //exit debugger
...
Breakpoint
._iput+000000     stmw    r29,FFFFFFF4(stkp)  <2FF3B24C> r29=09D75BD0,FFFFFFF4(stkp)=2FF3B24C
KDB(3)> gt @lr  //go to the link register value
.jfs_rele+0000B8 (sid:00000000) step < ctx: thread+001680 >
...
.jfs_rele+0000B8        b    <.jfs_rele+00007C>  r3=0
KDB(1)>
```

## n, s, S, and B subcommand

### Purpose

The **n**, **s**, **S** and **B** subcommands provide step functions.

**Note:** These subcommands are only available within the KDB kernel debugger. They are not included in the **kdb** command.

### Syntax

**n** [*count*]

**s** [*count*]

**S** [*count*]

**B** [*count*]

### Parameters

- *count* – Specifies the number of times the subcommand runs.

**n** – Runs the number of instructions specified by *count*, but it treats subroutine calls as a single instruction. If specified without a number, it runs one instruction.

**s** – Runs the number of instructions specified by the *count* parameter.

**S** – Runs instructions until it encounters a **bl** or **br** branch instruction. If the *count* parameter is used, the number specifies how many **bl** and **br** instructions are reached before the KDB Kernel Debugger stops.

**B** – Runs instructions until it encounters any branch instruction. If the *count* parameter is used, the number specifies how many branch instructions are reached before the KDB Kernel Debugger stops.

On POWER-based machines, steps are implemented with the **SE** bit of the **msr** status register of the processor. This bit is automatically associated with the thread or process context. The thread or process context can migrate from one processor to another.

You can interrupt any of these subcommands by pressing the Del key. Every time the KDB kernel debugger takes a step, it checks to see whether the Del key was pressed. This allows you to break into the KDB kernel debugger if the call is taking an inordinate amount of time.

If no intervening subcommands are run, any of the step commands can be repeated by pressing the Enter key.

Be aware that when you use these subcommands, an exception to the processor is made for each of the debugged program's instruction. One side-effect of exceptions is that it breaks reservations. The **stcwx** instruction cannot succeed if any breakpoint occurred after the last **larwx** instruction. The net effect is that you cannot use these subcommands with lock and atomic routines. If you do, you loop in the lock routine.

Some instructions are broken by exceptions. For example, **rfi** moves to and from **srr0 srr1**. The KDB kernel debugger tries to prevent this by printing a warning message.

When you want to take control of a sleeping thread, switch to the sleeping thread with the **sw** subcommand and type the **s** subcommand. The step is set inside the thread context, and when the thread runs again, the step breakpoint occurs.

## Aliases

The aliases are:

**n** – **nexti**
**s** – **stepi**

There are no aliases for the following:

**S**
**B**

## Example

The following is an example of how to use the **n**, **s**, and **B** subcommands:

```
KDB(1)> b .vno_close+00005C  //enable break point on vno_close+00005C
vno_close+00005C (sid:00000000) permanent & global
KDB(1)> e  //exit debugger
Breakpoint
.vno_close+00005C     lwz    r11,30(r4)          r11=0,30(r4)=xix_vops+000030
KDB(1)> s 10  //single step 10 instructions
.vno_close+000060     lwz    r5,68(stkp)         r5=FFD00000,68(stkp)=2FF97DD0
.vno_close+000064     lwz    r4,0(r5)            r4=xix_vops,0(r5)=file+0000C0
.vno_close+000068     lwz    r5,14(r5)           r5=file+0000C0,14(r5)=file+0000D4
.vno_close+00006C      bl    <._ptrgl>           r3=05AB620C
._ptrgl+000000     lwz    r0,0(r11)           r0=.closef+0000F4,0(r11)=xix_close
._ptrgl+000004     stw    toc,14(stkp)        toc=TOC,14(stkp)=2FF97D7C
._ptrgl+000008   mtctr    r0                  <.xix_close+000000>
._ptrgl+00000C     lwz    toc,4(r11)          toc=TOC,4(r11)=xix_close+000004
._ptrgl+000010     lwz    r11,8(r11)          r11=xix_close,8(r11)=xix_close+000008
._ptrgl+000014   bcctr                        <.xix_close>
KDB(1)> <CR/LF>  //repeat last single step command
.xix_close+000000    mflr    r0                  <.vno_close+000070>
.xix_close+000004     stw    r31,FFFFFFFC(stkp)  r31=_vno_fops$$,FFFFFFFC(stkp)=2FF97D64
.xix_close+000008     stw    r0,8(stkp)          r0=.vno_close+000070,8(stkp)=2FF97D70
.xix_close+00000C    stwu    stkp,FFFFFFA0(stkp) stkp=2FF97D68,FFFFFFA0(stkp)=2FF97D08
.xix_close+000010     lwz    r31,12B8(toc)       r31=_vno_fops$$,12B8(toc)=_xix_close$$
.xix_close+000014     stw    r3,78(stkp)         r3=05AB620C,78(stkp)=2FF97D80
.xix_close+000018     stw    r4,7C(stkp)         r4=00000020,7C(stkp)=2FF97D84
.xix_close+00001C     lwz    r3,12BC(toc)        r3=05AB620C,12BC(toc)=xclosedbg
.xix_close+000020     lwz    r3,0(r3)            r3=xclosedbg,0(r3)=xclosedbg
.xix_close+000024     lwz    r4,12C0(toc)        r4=00000020,12C0(toc)=pfsdbg
KDB(1)> r  //return to the end of function
.vno_close+000070     lwz    toc,14(stkp)        toc=TOC,14(stkp)=2FF97D7C
KDB(1)> S 4  //return to the end of function
.vno_close+000088      bl    <._ptrgl>           r3=05AB620C
.xix_rele+00010C      bl    <.vn_free>          r3=05AB620C
.vn_free+000140      bl    <.gpai_free>        r3=gpa_vnode
.gpai_free+00002C      br                        <.vn_free+000144>
KDB(1)> <CR/LF>  //repeat last command
.vn_free+00015C      br                        <.xix_rele+000110>
.xix_rele+000118      bl    <.iput>             r3=058F9360
.iput+0000A4      bl    <.iclose>           r3=058F9360
.iclose+000148      br                        <.iput+0000A8>
KDB(1)> <CR/LF>  //repeat last command
.iput+0001A4      bl    <.insque2>          r3=058F9360
.insque2+00004C      br                        <.iput+0001A8>
.iput+0001D0      br                        <.xix_rele+00011C>
.xix_rele+000164      br                        <.vno_close+00008C>
KDB(1)> r  //return to the end of function
.vno_close+00008C     lwz    toc,14(stkp)        toc=TOC,14(stkp)=2FF97D7C
KDB(1)>
```

# Chapter 18. Debugger trace points subcommands

**Note:** Debugger trace points subcommands are specific to the KDB kernel debugger. They are not available in the **kdb** command.

The subcommands in this category are used to trace the running of a specified address and stop KDB kernel debugger based on conditions. These subcommands include the following:

- bt
- test
- ct
- cat

## bt subcommand

### Purpose

The **bt** subcommand traces each a specified address each time it is run.

**Note:** This subcommand is only available within the KDB kernel debugger. It is not included in the **kdb** command.

### Syntax

**bt** [**-p** | **-v**] [*address* [*script*]]

### Parameters

- **-p** – Indicates that the trace address is a physical or real address.
- **-v** – Indicates that the trace address is an effective or virtual address.
- *address* – Specifies the address of the trace point. Symbols, hexadecimal values, or hexadecimal expressions can be used to specify an address.
- *script* – Lists subcommands to be run each time the indicated trace point is run. The script is delimited by quote (″) characters and commands within the script are delimited by semicolons (;).

Each time a trace point is encountered, a message is displayed indicating that the trace point was encountered. The displayed message indicates the first entry from the stack. However, this can be changed by using the script parameter.

If the **bt** subcommand is invoked with no parameters, the current list of break and trace points is displayed. The number of combined active trace and break points is limited to 32.

It is possible to specify whether the trace address is a physical or a virtual address with the **-p** and **-v** options respectively. By default, the KDB kernel debugger chooses the current state of the machine. If the subcommand is entered before VMM initialization, the address is the physical or real address. If the subcommand is entered after VMM initialization, the address is the effective or virtual address.

The **segment id** (**sid**) is always used to identify a trace point because effective or virtual addresses can have multiple translations in several virtual spaces. When debugging is resumed after a trace point is encountered, **kdb** must reinstall the correct instruction. During this time (one step if no interrupt is encountered), it is possible to miss the trace on other processors.

The script parameter allows a set of **kdb** subcommands to run when a trace point is reached. The set of subcommands comprising the script must be delimited by double quote characters (″). Individual subcommands within the script must be ended by a semicolon (;). One of the most useful subcommands that can be used in a script is the "test subcommand" on page 130. If this subcommand is included in the script, each time the trace point is reached the condition of the **test** subcommand is checked by the KDB kernel debugger. If the condition is true, a break occurs.

### Aliases

No aliases.

### Example

The following is an example of how to use the **bt** subcommand:

```
KDB(0)> bt open  //enable trace on open()
KDB(0)> bt  //display current active traces
0:      .open+000000 (sid:00000000)  trace {hit: 0}
KDB(0)> e  //exit debugger
```

```
...
open+00000000 (2FF7FF2B, 00000000, DEADBEEF)
open+00000000 (2FF7FF2F, 00000000, DEADBEEF)
open+00000000 (2FF7FF33, 00000000, DEADBEEF)
open+00000000 (2FF7FF37, 00000000, DEADBEEF)
open+00000000 (2FF7FF3B, 00000000, DEADBEEF)
...
KDB(0)> bt  //display current active traces
0:      .open+000000 (sid:00000000)  trace {hit: 5}
KDB(0)>
```

Open routine is traced with a script to display **iar** and **lr** registers and to show what is pointed to by the first parameter (r3).

```
KDB(0)> bt open "dr iar; dr lr; d @r3"  //enable trace on open()
KDB(0)> bt  //display current active traces
0:      .open+000000 (sid:00000000)  trace {hit: 0} {script: dr iar; dr lr;d @r3}
KDB(0)> e  //exit debugger
iar : 001C5BA0
.open+000000    mflr    r0                 <.svc_flih+00011C>
lr  : 00003B34
.svc_flih+00011C     lwz    toc,4108(0)         toc=TOC,4108=g_toc
2FF7FF3F: 7362 696E  0074 6D70  0074 6F74  6F00 7500   sbin.tmp.toto.u.
KDB(0)> bt  //display current active traces
0:      .open+000000 (sid:00000000)  trace {hit: 1} {script: dr iar; dr lr;d @r3}
KDB(0)> ct open  //clear trace on open
KDB(0)>
```

This example shows how to trace and stop when a condition is true. For example, when global data is greater than the specified value, and 923 hits were necessary to reach this condition.

```
KDB(0)> bt sys_timer "[ @time >= 2b8c8c00 ] "  //enable trace on sys_timer()
KDB(0)> e  //exit debugger
...
Enter kdb [ @time >= 2b8c8c00 ]
KDB(0) bt  //display current active traces
0:      .sys_timer+000000 (sid:00000000)  trace {hit: 923} {script: [ @time >= 2b8c8c00 ] }
KDB(0)> cat  //clear all traces
```

# test subcommand

## Purpose

The **test** subcommand can be used in conjunction with the "bt subcommand" on page 128 to break at a specified address when a condition becomes true.

## Syntax

**test** *cond*

## Parameters

- *cond* – Specifies the conditional expression that evaluates to a value of either true or false.

The conditional test requires two operands and a single operator. Operands include symbols, hexadecimal values, and hexadecimal expressions. Comparison operators that are supported include: ==, !=, >=, <=, >, and <. Additionally, the bitwise operators ^ (exclusive OR), & (AND), and | (OR) are supported. When bitwise operators are used, any non-zero result is considered to be true.

The syntax for the **test** subcommand requires that the operands and operator be delimited by spaces. This is very important to remember if the left square bracket ( [ ) alias is used. For example, the subcommand `test kernel_heap != 0` can be written as `[ kernel_heap != 0` . However, this subcommand is not valid if `kernel_heap`, `!=`, and `0` were not preceded by and followed by spaces.

## Aliases

**[**

## Example

The following is an example of how to use the **[** alias for the **test** subcommand:

```
KDB(0)> bt open "[ @sysinfo >= 3d ]"  //stop on open() if condition true
KDB(0)> e  //exit debugger
...
Enter kdb [ @sysinfo >= 3d ]
KDB(1)> bt  //display current active trace break points
0:      .open+000000 (sid:00000000)  trace {hit: 1} {script: [ @sysinfo >= 3d ]}
KDB(1)> dw sysinfo 1  //print sysinfo value
sysinfo+000000: 0000004A
```

# cat and ct subcommands

## Purpose

The **cat** subcommand erases all trace points. The **ct** subcommand erases individual trace points.

## Syntax

**cat**

**ct** *slot* | [**-p** | **-v**] *Address*

## Parameters

- *slot* – Identifies the slot number for a trace point. This parameter must be a decimal value.
- **-p** – Indicates the trace address is a physical or real address.
- **-v** – Indicate the trace address is an effective or virtual address.
- *Address* – Identifies the address of the trace point. Symbols, hexadecimal values, or hexadecimal expressions can be used to specify an address.

You can specify the trace point cleared by the **ct** subcommand by a slot number or by an address. By default, KDB kernel debugger chooses the current state of the machine. If the subcommand is entered before VMM initialization, the address is the physical or real address. If the subcommand is entered after VMM initialization, the address is the effective or virtual address.

**Note:** Slot numbers are not fixed. To clear slot 1 and slot 2 type `ct 2; ct 1` or type`ct 1; ct 1`. Do not type `ct 1; ct 2`.

## Aliases

No aliases.

## Example

The following is an example of how to use the **cat** and the **ct** subcommands:

```
KDB(0)> bt open  //enable trace on open()
KDB(0)> bt close  //enable trace on close()
KDB(0)> bt readlink  //enable trace on readlink()
KDB(0)> bt  //display current active traces
0:      .open+000000 (sid:00000000)  trace {hit: 0}
1:      .close+000000 (sid:00000000)  trace {hit: 0}
2:      .readlink+000000 (sid:00000000)  trace {hit: 0}
KDB(0)> ct 1  //clear trace slot 1
KDB(0)> bt  //display current active traces
0:      .open+000000 (sid:00000000)  trace {hit: 0}
1:      .readlink+000000 (sid:00000000)  trace {hit: 0}
KDB(0)> cat  //clear all active traces
KDB(0)> bt  //display current active traces
No breakpoints are set.
KDB(0)>
```

# Chapter 19. Watch DABR subcommands

The subcommands in this category are used to enter the debugger on a load or store instruction. These subcommands include the following:

- wr
- ww
- wrw
- cw
- lwr
- lww
- lwrw
- lcw

## wr, ww, wrw, cw, lwr, lww, lwrw, and lcw subcommands

### Purpose

The **wr** subcommand stops on a load instruction. The **ww** subcommand stops on a store instruction. The **wrw** subcommand stops either on a load or a store instruction.

The **cw** subcommand clears the last watch subcommand. The **lwr**, **lww**, **lwrw**, and **lcw** subcommands allow you to establish a watchpoint for a specific processor.

**Note:** These subcommands are only available within the KDB kernel debugger. They are not included in the **kdb** command.

### Syntax

**wr** [[**-e** | **-p** | **-v**] *address* [*size*]]

**ww** [[**-e** | **-p** | **-v**] *address* [*size*]]

**wrw**[[**-e** | **-p** | **-v**] *address* [*size*]]

**cw**

**lwr**[[**-e** | **-p** | **-v**] *address* [*size*]]

**lww** [[**-e** | **-p** | **-v**] *address* [*size*]]

**lwrw** [[**-e** | **-p** | **-v**] *address* [*size*]]

**lcw**

### Parameters

- **-e** – Indicates that the address parameter is an effective or virtual address.
- **-p** – Indicates that the address parameter is a physical or real address.
- **-v** – Indicates that the address parameter is a virtual or effective address.
- *address* – Specifies the address to watch. Symbols, hexadecimal values, or hexadecimal expressions can be used to specify the address. If the address type is not specified, it is assumed to be an effective address.
- *size* – Indicates the number of bytes to watch. This parameter is a decimal value.

A watch register can be used on the Data Address Breakpoint Register (DABR) or HID5 on PowerPC 601 RISC Microprocessor to enter KDB kernel debugger when a specified effective address is accessed. The register holds a double-word effective address and bits to specify load and store operations.

With no parameter, the **wr**, **ww** and **wrw** subcommands print the current active watch subcommand.

The **wr**, **ww**, **wrw** and **cw** subcommands are global to all processors. The **lwr**, **lww**, **lwrw** and **lcw** subcommands are local. If no size is specified, the default size is 8 bytes and the address is double-word aligned. If a size is specified, KDB kernel debugger checks the faulting address with the specified range. If no match is found, KDB kernel debugger continues to run.

## Aliases

**wr** – **stop-r**

**ww** – **stop-w**

**wrw** – **stop-rw**

**cw** – **stop-cl**

**lwr** – **lstop-r**

**lww** – **lstop-w**

**lwrw** – **lstop-rw**

**lcw** – **lstop-cl**

## Example

The following is an example of how to use the **ww**, the **wr** and the **cw** subcommands:

```
KDB(0)> ww -p emulate_count  //set a data break point (physical address, write mode)
KDB(0)> ww  //print current data break points
CPU 0: emulate_count+000000 paddr=00238360 size=8 hit=0 mode=W
CPU 1: emulate_count+000000 paddr=00238360 size=8 hit=0 mode=W
KDB(0)> e  //exit the debugger
...
Watch trap: 00238360 <emulate_count+000000>
power_asm_emulate+00013C     stw    r28,0(r30)          r28=0000003A,0(r30)=emulate_count
KDB(0)> ww  //print current data break points
CPU 0: emulate_count+000000 paddr=00238360 size=8 hit=1 mode=W
CPU 1: emulate_count+000000 paddr=00238360 size=8 hit=0 mode=W
KDB(0)> wr sysinfo  //set a data break point (read mode)
KDB(0)> wr  //print current data break points
CPU 0: sysinfo+000000 eaddr=003BA9D0 vsid=00000000 size=8 hit=0 mode=R
CPU 1: sysinfo+000000 eaddr=003BA9D0 vsid=00000000 size=8 hit=0 mode=R
KDB(0)> e  //exit the debugger
...
Watch trap: 003BA9D4 <sysinfo+000004>
.fetch_and_add+000008   lwarx    r3,0,r6            r3=sysinfo+000004,r6=sysinfo+000004
KDB(0)> cw  //clear data break points
```

# Chapter 20. Branch target subcommands

The subcommands in this category provide access on some POWER-based platform processors for target address comparison and trap functions. These subcommands include the following:

- btac
- cbtac
- lbtac
- lcbtac

 

# btac, cbtac, lbtac, lcbtac subcommands

## Purpose

Some POWER-based platform processors support an optional branch target address comparison and trap feature. When available, this facility allows for a branch target comparison with some user-provided value with a trap to a specific interrupt vector upon a match. The KDB kernel debugger **btac**, **cbtac**, **lbtac**, and **lcbtac** subcommands provide access to this facility when it is present. The **btac** subcommand stops when Branch Target Address Compare (BTAC) is true. The **cbtac** subcommand clears the last **btac** subcommand. The **cbtac** subcommand is global to all processors. Each processor can have different addresses specified or cleared using the local **lbtac** and **lcbtac** subcommands.

**Note:** These subcommands are only available in the KDB kernel debugger. They are not included in the **kdb** command.

## Syntax

**btac** [ [**-e** | **-p** | **-v**] *address*]

**cbtac**

**lbtac** [ [**-e** | **-p** | **-v**] *address*]

**lcbtac**

## Parameters

- **-p** – Indicates that the *address* parameter is considered to be a physical or real address.
- **-v** – Indicates that the *address* parameter is considered to be a virtual or effective address.
- **-e** – Indicates that the *address* parameter is considered to be an effective or virtual address.
- *address* – Specifies the address of the branch target. Symbols, hexadecimal values, or hexadecimal expressions can be used to specify the address.

The flags are mutually exclusive. The default flag is **-e**.

## Aliases

No aliases.

## Example

The following is an example of how to use the **btac**, the **lbtac** and the **cbtac** subcommands:

```
KDB(7)> btac open  //set BRAT on open function
KDB(7)> btac  //display current BRAT status
CPU 0: .open+000000 eaddr=001B5354 vsid=00000000 hit=0
CPU 1: .open+000000 eaddr=001B5354 vsid=00000000 hit=0
CPU 2: .open+000000 eaddr=001B5354 vsid=00000000 hit=0
CPU 3: .open+000000 eaddr=001B5354 vsid=00000000 hit=0
CPU 4: .open+000000 eaddr=001B5354 vsid=00000000 hit=0
CPU 5: .open+000000 eaddr=001B5354 vsid=00000000 hit=0
CPU 6: .open+000000 eaddr=001B5354 vsid=00000000 hit=0
CPU 7: .open+000000 eaddr=001B5354 vsid=00000000 hit=0
KDB(7)> e  //exit the debugger
...
Branch trap: 001B5354 <.open+000000>
.sys_call+000000  bcctrl                     <.open>
KDB(5)> btac  //display current BRAT status
CPU 0: .open+000000 eaddr=001B5354 vsid=00000000 hit=0
CPU 1: .open+000000 eaddr=001B5354 vsid=00000000 hit=0
CPU 2: .open+000000 eaddr=001B5354 vsid=00000000 hit=0
```

```
CPU 3: .open+000000 eaddr=001B5354 vsid=00000000 hit=0
CPU 4: .open+000000 eaddr=001B5354 vsid=00000000 hit=0
CPU 5: .open+000000 eaddr=001B5354 vsid=00000000 hit=1
CPU 6: .open+000000 eaddr=001B5354 vsid=00000000 hit=0
CPU 7: .open+000000 eaddr=001B5354 vsid=00000000 hit=0
KDB(5)> lbtac close  //set local BRAT on close function
KDB(5)> e  //exit the debugger
...
Branch trap: 001B5354 <.open+000000>
.sys_call+000000  bcctrl                        <.open>
KDB(7)> e  //exit the debugger
...
Branch trap: 00197D40 <.close+000000>
.sys_call+000000  bcctrl                        <.close>
KDB(5)> e  //exit the debugger   ...
Branch trap: 001B5354 <.open+000000>
.sys_call+000000  bcctrl                        <.open>
KDB(6)> btac  //display current BRAT status
CPU 0: .open+000000 eaddr=001B5354 vsid=00000000 hit=0
CPU 1: .open+000000 eaddr=001B5354 vsid=00000000 hit=0
CPU 2: .open+000000 eaddr=001B5354 vsid=00000000 hit=0
CPU 3: .open+000000 eaddr=001B5354 vsid=00000000 hit=0
CPU 4: .open+000000 eaddr=001B5354 vsid=00000000 hit=0
CPU 5: .close+000000 eaddr=00197D40 vsid=00000000 hit=1
CPU 6: .open+000000 eaddr=001B5354 vsid=00000000 hit=1
CPU 7: .open+000000 eaddr=001B5354 vsid=00000000 hit=1
KDB(6)> cbtac  //reset all BRAT registers
```

# Chapter 21. Namelist and symbols subcommands

The subcommands in this category are used to change namelists and symbols. These subcommands include the following:

- nm
- ts
- ns
- which

 

# nm and ts subcommands

## Purpose

The **nm** subcommand translates symbols to addresses. The **ts** subcommand translates addresses to symbolic representations.

## Syntax

**nm** *symbol*

**ts** *effectiveaddress*

## Parameters

- *symbol* – Specifies the symbol name.
- *effectiveaddress* – Specifies the effective address to be translated. This parameter can be a hexadecimal value or an expression.

## Aliases

No aliases.

## Example

The following is an example of how to use the **nm** and the **ts** subcommands:

```
KDB(0)> nm __ublock  //print symbol value
Symbol Address : 2FF3B400
KDB(0)> ts E3000000  //print symbol name
proc+000000
```

# ns subcommand

## Purpose

The **ns** subcommand toggles symbolic name translation on and off.

## Syntax

**ns**

## Parameters

No parameters.

## Aliases

No aliases.

## Example

The following is an example of how to use the **ns** subcommand:

```
KDB(0)> dc d000 5 //display code at address D000
   ___memcmp+000000     cmpw    cr1,r3,r4
   ___memcmp+000004    srwi.    r12,r5,2
   ___memcmp+000008   clrlwi    r11,r5,1E
   ___memcmp+00000C       li    r7,0
   ___memcmp+000010     beq-    cr1.eq,<__memcmp+000050>
   KDB(0)> ns //disable symbol printing
   Symbolic name translation off
   KDB(0)> dc d000 5 //display code at address D000
   0000D000     cmpw    cr1,r3,r4
   0000D004    srwi.    r12,r5,2
   0000D008   clrlwi    r11,r5,1E
   0000D00C       li    r7,0
   0000D010     beq-    cr1.eq,<0000D050>
   KDB(0)> ns //enable symbol printing
   Symbolic name translation on
   KDB(0)>
```

## which subcommand

### Purpose

The **which** subcommand displays the name of the kernel source file that contains the *address*.

**Note:** The **which** subcommand is only available in the **kdb** command.

### Syntax

**which** | *address*

### Parameters

- *address* – Locates the kernel source file that contains the symbol at the specified address and displays the following:
  - The symbol corresponding to the address
  - The start address of the symbol
  - The kernel source file name containing the symbol

### Aliases

**wf**

### Example

The following is an example of how to use the **which** subcommand:

```
> which main
   Addr: 0022A700  Symbol: .main
   Name: ../../../../../src/bos/kernel/si/main.c
```

# Chapter 22. PCI configuration space and I/O debugging subcommands

The subcommands in this category are used to debug I/O errors and PCI configuration space errors. These subcommands include the following:

- dpcib
- dpcih
- dpciw
- mpcib
- mpcih
- mpciw
- buserr
- businfo

# dpcib, dpcih, and dpciw subcommand

## Purpose

The **dpcib** (display PCI byte), **dpcih** (display PCI halfword), and **dpciw** (display PCI word) subcommands read data from the PCI Configuration Space.

## Syntax

**dpcib** *Bid PCIslot RegOffset*

**dpcih** *Bid PCIslot RegOffset*

**dpciw** *Bid PCIslot RegOffset*

## Parameters

- *Bid* – Identifies the Bus Identifier of the PCI bus.
- *PCIslot* – Combines the device number on the PCI bus and the function number on that PCI slot. The combination uses the following formula:

  ```
  PCIslot = (device_num * 8) + function
  ```
- *RegOffset* – Identifies a zero-based byte offset of the register to read in a PCI Configuration Space.

## Aliases

No aliases.

## Example

The following is an example of how to use the **dpcib**, the **dpcih**, and **dpciw** subcommands:

```
KDB(0)> businfo                       //get PCI bus id
      **********   PCI BUSES   ***********
      ADDRESS   BID       BUS_NUM   PHB_UNIT_ID       REGIONS
      30043400  900000C0  00000000  00000000FEF00000  00000004
      30043500  900000C1  00000040  00000000FEE00000  00000002
      **********   OTHER BUSES   ***********
      ADDRESS   BID       BUS_NUM   PHB_UNIT_ID       REGIONS
      00459AE0  90000040  00000000  0000000000000000  00000001
      00459F60  90000100  00000000  0000000000000000  00000002
      0045AB60  90000300  00000000  0000000000000000  00000001
      KDB(0)> dpcib 900000c0 01 4           //display byte of data
      00000104:  46
      KDB(0)> dpcih 900000c0 01 4           //display halfword of data
      00000104:  4600
      KDB(0)> dpciw 900000c0 01 4           //display word of data
      00000104:  46008022
```

## mpcib, mpcih, and mpciw subcommands

### Purpose

The **mpcib** (modify PCI byte), **mpcih** (modify PCI halfword), and **mpciw** (modify PCI word) subcommands write data to the PCI Configuration Space.

### Syntax

**mpcib** *Bid PCIslot RegOffset*

**mpcih** *Bid PCIslot RegOffset*

**mpciw** *Bid PCIslot RegOffset*

### Parameters

- *Bid* – Identifies the Bus Identifier of the PCI bus.
- *PCIslot* – Combines the device number on the PCI bus and the function number on that PCI slot. The combinations uses the following formula:

  ```
  PCIslot = (device_num * 8) + function
  ```
- *RegOffset* – Identifies a zero-based byte offset of the register to read in a PCI Configuration Space.

These commands are interactive and each modification is entered one-by-one. The first unexpected input stops modification. A period (.), for example, can be used to indicate the end of the data.

### Aliases

No aliases.

### Example

The following is an example of how to use the **mpcib**, the **mpcih**, and the **mpciw** subcommands:

```
KDB(0)> businfo                        //get PCI bus id
      **********   PCI BUSES   ***********
      ADDRESS   BID       BUS_NUM   PHB_UNIT_ID       REGIONS
      30043400  900000C0  00000000  00000000FEF00000  00000004
      30043500  900000C1  00000040  00000000FEE00000  00000002
      **********   OTHER BUSES   ***********
      ADDRESS   BID       BUS_NUM   PHB_UNIT_ID       REGIONS
      00459AE0  90000040  00000000  0000000000000000  00000001
      00459F60  90000100  00000000  0000000000000000  00000002
      0045AB60  90000300  00000000  0000000000000000  00000001
      KDB(0)> dpciw 900000c0 80 10           //display word of data
      00008010:  01F0FF00
      KDB(0)> mpciw 900000c0 80 10           //modify word
      00008010:  01F0FF00  = ffffffff
      00008014:  00A010C0  = .
      KDB(0)> dpciw 900000c0 80 10           //display new word of data
      00008010:  E1FFFFFF
      KDB(0)> mpciw 900000c0 80 10           (reset word)
      00008010:  E1FFFFFF  = 01F0FF00
      00008014:  00A010C0  = .
      KDB(0)> dpciw 900000c0 80 10           //display reset word
      00008010:  01F0FF00
      KDB(0)> mpcib 900000c0 80 10           //modify specifying bytes
      00008010:  01  = ff
      00008011:  F0  = ff
      00008012:  FF  = ff
      00008013:  00  = ff
      00008014:  00  = .
      KDB(0)> dpciw 900000c0 80 10           //display new word of data
```

```
00008010:  E1FFFFFF
KDB(0)> mpciw 900000c0 80 10           //reset word
00008010:  E1FFFFFF  = 01F0FF00
00008014:  00A010C0  = .
KDB(0)> dpciw 900000c0 80 10           //display reset word
00008010:  01F0FF00
KDB(0)> mpcih 900000c0 80 10           //modify specifying halfwords
00008010:  01F0  = ffff
00008012:  FF00  = ffff
00008014:  00A0  = .
KDB(0)> dpciw 900000c0 80 10           //display new word of data
00008010:  E1FFFFFF
KDB(0)> mpciw 900000c0 80 10           //reset word
00008010:  E1FFFFFF  = 01F0FF00
00008014:  00A010C0  = .
KDB(0)> dpciw 900000c0 80 10           //display reset word
00008010:  01F0FF00
KDB(0)>
```

## buserr subcommand

### Purpose

The **buserr** subcommand allows PCI bus error injection and manual exercise of EEH capabilities on a PCI slot.

### Syntax

**buserr bid slot** [*operation*] [*function*] [*bus_addr*]

### Parameters

- **bid** – Specifies the bus id. It must be a hexadecimal value.
- **slot** – Specifies the slot number. It must be a hexadecimal value.
- *operation* – Specifies the operation code. Accepted values are:
  - 1 - Query slot capabilities and slot state. Displays the state of a slot and information about whether EEH is supported by the slot.
  - 2 - Set slot state. Allows enabling or disabling EEH on a slot or enabling load, store or DMA operation.
  - 3 - Inject a bus error. Performs error injection on a specified bus and slot at a given bus address. The errors can be injected in either memory, I/O or configuration address spaces of a PCI bus. Also, the errors can be on a load or store operation.
  - 4 - Reset slot. This is a way to recover from an EEH event. This operation code can be used to assert and deassert the reset signal on the bus. The reset signal should be asserted for at least 100 milliseconds before deasserting it.
  - 5 - Configure PCI bridge on the adapter. Allows the bridge on an adapter to be configured following a slot reset. This is a required step in complete error recovery for the bridged-adapters such as Ethernet cards.
- *function* – Specifies the function code. It must be a hexadecimal value. Function codes are dependent on the operation code. The available function codes are:
  - Operation code 1 - Query slot capabilities and slot state. There are no function codes available.
  - Operation code 2 - Set slot state:
    - 0 - Disable EEH
    - 1 - Enable EEH
    - 2 - Enable load/store
    - 3 - Enable DMA
  - Operation code 3 - Inject a bus error:
    - 0 - Load to PCI Memory Address Space - inject an Address Parity Error
    - 1 - Load to PCI Memory Address Space - inject a Data Parity Error
    - 2 - Load to PCI I/O Address Space - inject an Address Parity Error
    - 3 - Load to PCI I/O Address Space - inject a Data Parity Error
    - 4 - Load to PCI Configuration Space - inject an Address Parity Error
    - 5 - Load to PCI Configuration Space - inject a Data Parity Error
    - 6 - Store to PCI Memory Address Space - inject an Address Parity Error
    - 7 - Store to PCI Memory Address Space - inject a Data Parity Error
    - 8 - Store to PCI I/O Address Space - inject an Address Parity Error
    - 9 - Store to PCI I/O Address Space - inject a Data Parity Error
    - A - Store to PCI Configuration Space - inject an Address Parity Error
    - B - Store to PCI Configuration Space - inject a Data Parity Error

- C - DMA read to PCI Memory Address Space - inject an Address Parity Error
- D - DMA read to PCI Memory Address Space - inject a Data Parity Error
- E - DMA read to PCI Memory Address Space - inject a Master Abort Error
- F - DMA read to PCI Memory Address Space - inject a Target Abort Error
- 10 - DMA write to PCI Memory Address Space - inject an Address Parity Error
- 11 - DMA write to PCI Memory Address Space - inject a Data Parity Error
- 12 - DMA write to PCI Memory Address Space - inject a Master Abort Error
- 13 - DMA write to PCI Memory Address Space - inject a Target Abort Error

– For operation code 4 - Reset slot:
  - 0 - Deactivate Reset
  - 1 - Activate Reset
– For operation code 5 - Configure PCI bridge on the adapter. There are no function codes available.

• *bus_addr* – Specifies the bus address. *bus_addr* is only used with operation code 3 - Inject a bus error. *bus_addr* must be a hexadecimal value.

## Aliases

No aliases.

## Example

The following is an example of how to use the **buserr** subcommand:

```
KDB(0)> buserr 900000d5 8 1          //query state of slot and if EEH supported

Query Slot Capabilities And Slot State
--------------------------------------
Reset State: Reset deactive, EEH not stopped
Slot Capabilities: EEH supported
Success

Select an Operation Code
 1) Query Slot Capabilities And Slot State
 2) Set Slot State
 3) Inject a bus error
 4) Reset Slot
 5) Configure PCI Bridge on the Adapter
 99) Exit

 Enter you choice: 99
KDB(0)> buserr 900000d5 8 4 1          //assert reset

Reset Slot
----------
Success

Select an Operation Code
 1) Query Slot Capabilities And Slot State
 2) Set Slot State
 3) Inject a bus error
 4) Reset Slot
 5) Configure PCI Bridge on the Adapter
 99) Exit

 Enter you choice: 99
KDB(0)> buserr 900000d5 8 4 0          //deassert reset

Reset Slot
----------
Success
```

```
Select an Operation Code
 1) Query Slot Capabilities And Slot State
 2) Set Slot State
 3) Inject a bus error
 4) Reset Slot
 5) Configure PCI Bridge on the Adapter
 99) Exit

 Enter you choice: 99
KDB(0)> buserr 900000d5 8 3 0 0xf8000000     //inject an address parity error

Inject a bus error
------------------
Success

Select an Operation Code
 1) Query Slot Capabilities And Slot State
 2) Set Slot State
 3) Inject a bus error
 4) Reset Slot
 5) Configure PCI Bridge on the Adapter
 99) Exit

 Enter your choice: 99
```

# businfo subcommand

## Purpose

The **businfo** subcommand displays information about all registered buses or about a specified bus.

## Syntax

**businfo** [**-a** | **-b** *Bid* | *eaddr* ]

## Parameters

- **-a** – Displays data for all valid buses.
- **-b** *Bid* – Displays data for bus specified by bus id *Bid*.
- *eaddr* – Displays data for the bus at this address.

## Aliases

No aliases.

## Example

The following is an example of how to use the **businfo** subcommand:

```
KDB(0)> businfo          //display summary
**********   PCI BUSES   ***********
ADDRESS   BID       BUS_NUM   PHB_UNIT_ID       REGIONS
30043400  900000C0  00000000  00000000FEF00000  00000004
30043500  900000C1  00000040  00000000FEE00000  00000002
**********   OTHER BUSES   ***********
ADDRESS   BID       BUS_NUM   PHB_UNIT_ID       REGIONS
00459AE0  90000040  00000000  0000000000000000  00000001
00459F60  90000100  00000000  0000000000000000  00000002
0045AB60  90000300  00000000  0000000000000000  00000001
KDB(0)> businfo -b 900000C0          //display details specifying bus id
next = 00000000                bid = 900000C0
d_map_init = 021D4B08          disable_io = 00000000
num_regions = 00000004
ioaddr[0] = 00000000F8000000    ioaddr[1] = 0000000000000000
ioaddr[2] = 0000000000000000    ioaddr[3] = 00000000CF000000
ioaddr[4] = 0000000000000000    ioaddr[5] = 0000000000000000
ioaddr[6] = 0000000000000000    ioaddr[7] = 0000000000000000
ioaddr[8] = 0000000000000000    ioaddr[9] = 0000000000000000
ioaddr[10] = 0000000000000000    ioaddr[11] = 0000000000000000
ioaddr[12] = 0000000000000000    ioaddr[13] = 0000000000000000
ioaddr[14] = 0000000000000000    ioaddr[15] = 0000000000000000
bus_specific_data = 00000000    PHB_Unit_ID = 00000000FEF00000
bmap = 00000000
eeh_init = 021D4B14            eeh_init_multifunc = 021D4BD4
reserved3 = 00000000           reserved4 = 00000000

KDB(0)> businfo 00459AE0          //display details specifying address
next = 00000000                bid = 90000040
d_map_init = 00000000          disable_io = 00000000
num_regions = 00000001
ioaddr[0] = 0000000000000000    ioaddr[1] = 0000000000000000
ioaddr[2] = 0000000000000000    ioaddr[3] = 0000000000000000
ioaddr[4] = 0000000000000000    ioaddr[5] = 0000000000000000
ioaddr[6] = 0000000000000000    ioaddr[7] = 0000000000000000
ioaddr[8] = 0000000000000000    ioaddr[9] = 0000000000000000
ioaddr[10] = 0000000000000000    ioaddr[11] = 0000000000000000
ioaddr[12] = 0000000000000000    ioaddr[13] = 0000000000000000
ioaddr[14] = 0000000000000000    ioaddr[15] = 0000000000000000
bus_specific_data = 00000000    PHB_Unit_ID = 0000000000000000
bmap = 00000000
```

```
eeh_init = 00000000             eeh_init_multifunc = 00000000
reserved3 = 00000000            reserved4 = 00000000

KDB(0)> businfo -a          //display details for all valid buses
**********   PCI BUSES   ***********
Printing Hash bucket 00000000
-----------------------------
next = 00000000                 bid = 900000C0
d_map_init = 021D4B08           disable_io = 00000000
num_regions = 00000004
ioaddr[0] = 00000000F8000000    ioaddr[1] = 0000000000000000
ioaddr[2] = 0000000000000000    ioaddr[3] = 00000000CF000000
ioaddr[4] = 0000000000000000    ioaddr[5] = 0000000000000000
ioaddr[6] = 0000000000000000    ioaddr[7] = 0000000000000000
ioaddr[8] = 0000000000000000    ioaddr[9] = 0000000000000000
ioaddr[10] = 0000000000000000    ioaddr[11] = 0000000000000000
ioaddr[12] = 0000000000000000    ioaddr[13] = 0000000000000000
ioaddr[14] = 0000000000000000    ioaddr[15] = 0000000000000000
bus_specific_data = 00000000    PHB_Unit_ID = 00000000FEF00000
bmap = 00000000
eeh_init = 021D4B14             eeh_init_multifunc = 021D4BD4
reserved3 = 00000000            reserved4 = 00000000

Printing Hash bucket 00000001
-----------------------------
next = 00000000                 bid = 900000C1
(0)> more (^C to quit) ? ^C        //interrupt
```

# Chapter 23. Display kernel data structures subcommands

The subcommands in this category are used to print the **var** and **drvars** structure and the system configuration of a machine and to display information about IPL control blocks, interrupt handler tables and device switch tables. These subcommands include the following:

- var
- drvars
- ipl
- dev
- intr

## var subcommand

## Purpose

The **var** subcommand prints the **var** structure and the system configuration of the machine.

## Syntax

**var**

## Parameters

No parameters.

## Aliases

No aliases.

## Example

The following is an example of how to use the **var** subcommand:

```
KDB(7)> var  //print var information
var_hdr.var_vers..... 00000000 var_hdr.var_gen...... 00000045
var_hdr.var_size..... 00000030
v_iostrun............ 00000001 v_leastpriv.......... 00000000
v_autost............. 00000001 v_memscrub........... 00000000
v_maxup.............      200
v_bufhw.............       20 v_mbufhw............    32768
v_maxpout...........        0 v_minpout...........        0
v_clist.............    16384 v_fullcore........... 00000000
v_ncpus.............        8 v_ncpus_cfg..........        8
v_initlvl...........  0  0  0  0
v_lock..............      200 ve_lock.............. 00D3FA18 flox+003200
v_file..............     2303 ve_file.............. 0042EFE8 file+01AFD0
v_proc..............   131072 ve_proc.............. E305D000 proc+05D000
vb_proc.............. E3000000 proc+000000
v_thread............   262144 ve_thread............ E6046F80 thread+046F80
vb_thread............ E6000000 thread+000000

VMM Tunable Variables:

minfree.............      120 maxfree.............      128
minperm.............    12872 maxperm.............    51488
pfrsvdblks..........    13076
(7)> more (^C to quit) ? //continue
npswarn.............      512 npskill.............      128
minpgahead..........        2 maxpgahead..........        8
maxpdtblks..........        4 numsched............        4
htabscale............ FFFFFFFF aptscale............. 00000000
pd_npages............ 00080000

_SYSTEM_CONFIGURATION:

architecture..... 00000002 POWER_PC
implementation... 00000010 POWER_604
version.......... 00040004
width............ 00000020 ncpus............ 00000008
cache_attrib..... 00000001 CACHE separate I and D
icache_size...... 00004000 dcache_size...... 00004000
icache_asc....... 00000004 dcache_asc....... 00000004
icache_block..... 00000020 dcache_block..... 00000020
icache_line...... 00000040 dcache_line...... 00000040
L2_cache_size.... 00100000 L2_cache_asc..... 00000001
tlb_attrib....... 00000001 TLB separate I and D
itlb_size........ 00000040 dtlb_size........ 00000040
```

```
itlb_asc......... 00000002 dtlb_asc......... 00000002
priv_lck_cnt..... 00000000 prob_lck_cnt..... 00000000
resv_size........ 00000020 rtc_type......... 00000002
virt_alias....... 00000000 cach_cong........ 00000000
model_arch....... 00000001 model_impl....... 00000002
Xint............. 000000A0 Xfrac............ 00000003
```

# drvars subcommand

## Purpose

The **drvars** subcommand displays the global state of Dynamic Reconfiguration (DR) from the drvars structure, and displays state about any current DR operation from the **drparms** and **drvars** structures.

## Syntax

**drvars**

## Parameters

No parameters.

## Aliases

No aliases.

## Example

The following is an example of how to use the **drvars** subcommand:

```
KDB(0)> drvars
DRparms:

drp_operation..... 00000000
drp_op_idx........ FFFFFFFF
drp_phase......... FFFFFFFF
drp_errno......... 00000000
drp_secs.......... 00000000
drp_flags......... 00000000
drp_pid........... FFFFFFFF
drp_trb........... @ F10010F003FFC280 KERN_heap+3FFC280
drp_timeout....... @ 0000000003A63398 drparms+000028
drp_in............ @ 0000000003A633A8 drparms+000038
drp_apps_out...... @ 0000000003A63408 drparms+000098
drp_kx_out........ @ 0000000003A63430 drparms+0000C0

DRvars:

drbits............ 00000000
flags............. 00000000
lmb_addr.......... 0000000270000000
lmb_size.......... 0000000010000000
RMO_size.......... 0000000040000000
sys_lmbsize....... 0000000010000000
max_num_lmbs...... 00000024
actual_num_lmbs... 00000024
fixed_nfr......... 0000000000000000
dead_nfrs......... 00000000
lrudr_running..... 00
gencount.......... 0000000000000006
l_cpuX............ 00000000
l_cpuX_halted..... 00000000
l_cpuY............ 00000000
n_mpcs............ 00000000
gserver........... 00000000
server............ 00000000
trace............. 00000000
```

## ipl subcommand

### Purpose

The **ipl** subcommand displays information about IPL control blocks.

### Syntax

**ipl** [***** | **cpu** *index*]

### Parameters

- ***** – Displays summary information for all CPUs.
- **cpu** – Specifies the CPU number for the IPL control block to be displayed. The CPU is specified as a decimal value.
- *index* – Displays the specified index.

### Aliases

**iplcb**

### Example

The following is an example of how to use the **ipl** subcommand:

```
KDB(4)> ipl *  //print ipl control blocks
         INDEX  PHYS_ID INT_AREA ARCHITEC IMPLEMEN  VERSION

0038ECD0     0 00000000 FF100000 00000002 00000008 00010005
0038ED98     1 00000001 FF100080 00000002 00000008 00010005
0038EE60     2 00000002 FF100100 00000002 00000008 00010005
0038EF28     3 00000003 FF100180 00000002 00000008 00010005
0038EFF0     4 00000004 FF100200 00000002 00000008 00010005
0038F0B8     5 00000005 FF100280 00000002 00000008 00010005
0038F180     6 00000006 FF100300 00000002 00000008 00010005
0038F248     7 00000007 FF100380 00000002 00000008 00010005
KDB(4)> ipl //print current processor information

Processor Info 4 [0038EFF0]

num_of_structs.........00000008 index..................00000004
struct_size...........000000C8 per_buc_info_offset....0001D5D0
proc_int_area..........FF100200 proc_int_area_size.....00000010
processor_present......00000001 test_run..............0000006A
test_stat.............00000000 link..................00000000
link_address..........00000000 phys_id...............00000004
architecture..........00000002 implementation.........00000008
version...............00010005 width.................00000020
cache_attrib..........00000003 coherency_size.........00000020
resv_size.............00000020 icache_block...........00000020
dcache_block..........00000020 icache_size...........00008000
dcache_size...........00008000 icache_line...........00000040
dcache_line...........00000040 icache_asc............00000008
dcache_asc............00000008 L2_cache_size..........00100000
L2_cache_asc..........00000001 tlb_attrib............00000003
itlb_size.............00000100 dtlb_size.............00000100
itlb_asc..............00000002 dtlb_asc..............00000002
slb_attrib............00000000 islb_size.............00000000
dslb_size.............00000000 islb_asc..............00000000
(4)> more (^C to quit) ?  //continue
dslb_asc..............00000000 priv_lck_cnt..........00000000
prob_lck_cnt..........00000000 rtc_type..............00000001
rtcXint...............00000000 rtcXfrac..............00000000
busCfreq_HZ...........00000000 tbCfreq_HZ............00000000
```

```
System info [0038E534]
num_of_procs...........00000008 coherency_size.........00000020
resv_size..............00000020 arb_cr_addr............00000000
phys_id_reg_addr.......00000000 num_of_bsrr............00000000
bsrr_addr..............00000000 tod_type...............00000000
todr_addr..............FF0000C0 rsr_addr...............FF62006C
pksr_addr..............FF620064 prcr_addr..............FF620060
sssr_addr..............FF001000 sir_addr...............FF100000
scr_addr...............00000000 dscr_addr..............00000000
nvram_size.............00022000 nvram_addr.............FF600000
vpd_rom_addr...........00000000 ipl_rom_size...........00100000
ipl_rom_addr...........07F00000 g_mfrr_addr............FF107F80
g_tb_addr..............00000000 g_tb_type..............00000000
g_tb_mult..............00000000 SP_Error_Log_Table.....0001C000
pcccr_addr.............00000000 spocr_addr.............FF620068
pfeivr_addr............FF00100C access_id_waddr........00000000
loc_waddr..............00000000 access_id_raddr........00000000
(4)> more (^C to quit) ?  //continue
loc_raddr..............00000000 architecture...........00000001
implementation.........00000002 pkg_descriptor.........rs6ksmp
KDB(4)>
```

## devsw subcommand

### Purpose

The **devsw** subcommand displays device switch table entries.

### Syntax

**devsw** [*major* | *address*]

### Parameters

- *major* – Indicates the specific device switch table entry to be displayed by the major number. This is a hexadecimal value.
- *address* – Specifies the effective address of a driver. The device switch table entry with the driver closest to the indicated address is displayed. The specific driver is indicated. Symbols, hexadecimal values, or hexadecimal expressions can be used to specify the address.

### Aliases

**dev**

### Example

The following is an example of how to use the **dev** alias for the **devsw** subcommand:

```
KDB(0)> dev
Slot address 054F5040
MAJ#001  OPEN            CLOSE           READ            WRITE
         .syopen         .nulldev        .syread         .sywrite
         IOCTL           STRATEGY        TTY             SELECT
         .syioctl        .nodev          00000000        .syselect
         CONFIG          PRINT           DUMP            MPX
         .nodev          .nodev          .nodev          .nodev
         REVOKE          DSDPTR          SELPTR          OPTS
         .nodev          00000000        00000000        00000002

Slot address 054F5080
MAJ#002  OPEN            CLOSE           READ            WRITE
         .nulldev        .nulldev        .mmread         .mmwrite
         IOCTL           STRATEGY        TTY             SELECT
         .nodev          .nodev          00000000        .nodev
         CONFIG          PRINT           DUMP            MPX
         .nodev          .nodev          .nodev          .nodev
         REVOKE          DSDPTR          SELPTR          OPTS
         .nodev          00000000        00000000        00000002

(0)> more (^C to quit) ? ^C  //quit
KDB(0)> devsw 4  //device switch of major 0x4
Slot address 05640100
MAJ#004  OPEN            CLOSE           READ            WRITE
         .conopen        .conclose       .conread        .conwrite
         IOCTL           STRATEGY        TTY             SELECT
         .conioctl       .nodev          00000000        .conselect
         CONFIG          PRINT           DUMP            MPX
         .conconfig      .nodev          .nodev          .conmpx
         REVOKE          DSDPTR          SELPTR          OPTS
         .conrevoke      00000000        00000000        00000006
```

## intr subcommand

### Purpose

The **intr** subcommand prints a summary for entries in the interrupt handler table if no parameter or a slot number is entered.

### Syntax

**intr** [ *slot* | *address*]

### Parameters

- *slot* – Specifies the slot number in the interrupt handler table. This value must be a decimal value.
- *address* – Specifies the effective address of an interrupt handler. Symbols, hexadecimal values or hexadecimal expressions can be used to specify the address.

If no parameter is entered, the summary contains information for all entries. If a slot number is specified, only the selected entries are displayed. If an address parameter is entered, detailed information is displayed for the specified interrupt handler.

### Aliases

No aliases.

### Example

The following is an example of how to use the **intr** subcommand:

```
KDB(0)> intr  //interrupt handler table
             SLT INTRADDR HANDLER  TYPE LEVEL    PRIO BID      FLAGS

i_data+000068   1 055DF0A0 00000000 0000 00000003 0000 00000000 0000
i_data+000068   1 00364F88 00090584 0000 00000001 0000 00000000 0000
i_data+000068   1 003685B0 00090584 0001 00000008 0000 82000000 0000
i_data+000068   1 019E7D48 019E7BF0 0000 00000001 0000 820C0020 0010
i_data+0000E0  16 055DF060 00000000 0001 00000001 0000 82000080 0000
i_data+0000E0  16 00368718 000A24D8 0001 00000000 0000 82000080 0000
i_data+0000F0  18 055DF100 00000000 0001 00000000 0001 82080060 0010
i_data+0000F0  18 05B3BC00 01A55018 0001 00000002 0001 82080060 0010
i_data+000120  24 055DF0C0 00000000 0001 00000004 0000 82000000 0000
i_data+000120  24 003685B0 00090584 0001 00000008 0000 82000000 0000
i_data+000120  24 019E7D48 019E7BF0 0000 00000001 0000 820C0020 0010
i_data+000140  28 055DF160 00000000 0001 00000001 0003 820C0060 0010
i_data+000140  28 0A145000 01A741AC 0001 0000000C 0003 820C0060 0010
i_data+000150  30 055DF0E0 00000000 0001 00000000 0003 820C0020 0010
i_data+000150  30 055FC000 019E7AA8 0001 0000000E 0003 820C0020 0010
i_data+000160  32 055DF080 00000000 0001 00000002 0000 82100080 0000
i_data+000160  32 00368734 000A24D8 0001 00000000 0000 82100080 0000
i_data+0004E0 144 055DF020 00000000 0002 00000000 0000 00000000 0011
i_data+0004E0 144 00368560 000903B0 0002 00000002 0000 00000000 0011
i_data+000530 154 055DF040 00000000 0002 FFFFFFFF 000A 00000000 0011
i_data+000530 154 00368580 000903B0 0002 00000002 000A 00000000 0011
KDB(0)> intr 1  //interrupt handler slot 1
             SLT INTRADDR HANDLER  TYPE LEVEL    PRIO BID      FLAGS

i_data+000068   1 055DF0A0 00000000 0000 00000003 0000 00000000 0000
i_data+000068   1 00364F88 00090584 0000 00000001 0000 00000000 0000
i_data+000068   1 003685B0 00090584 0001 00000008 0000 82000000 0000
i_data+000068   1 019E7D48 019E7BF0 0000 00000001 0000 820C0020 0010
KDB(0)> intr 00368560  //interrupt handler address
addr.......... 00368560 handler....... 000903B0 i_hwassist_int+000000
bid........... 00000000 bus_type...... 00000002 PLANAR
```

```
next.......... 00000000 flags......... 00000011 NOT_SHARED MPSAFE
level......... 00000002 priority...... 00000000 INTMAX
i_count....... 00000014
KDB(0)>
```

# Chapter 24. Display VMM subcommands

The subcommands in this category can be used to display VMM information. These subcommands include the following:

- ames
- apt
- frameset
- free
- freelist
- ipc
- rtipc
- rtipcd
- lka
- lkh
- lkw
- mempool
- pdt
- pfhdata
- pft
- swhat
- pvt
- pta
- pte
- rmap
- rvsid
- scb
- segst64
- sr64
- ksp
- ste
- vmbufst
- vmaddr
- vmdmap
- vmint
- vmker
- vmlocks
- vmlog
- vmpool
- vmstat
- vmthrpgio
- vmwait
- vsidd
- vsidm
- zproc

- drlist
- drlist

## ames subcommand

### Purpose

The **ames** subcommand provides options for the display of the process address map for either the current process, a specified process, or a specified address map.

### Syntax

**ames** [*menu options*]

### Parameters

- *menu options* – Menu options and parameters can be entered along with the subcommand to avoid display of menus and prompts.

If this subcommand is invoked without arguments, menus and prompts are used to determine the data to be displayed. If the menu selections and required values are known they can be entered as subcommand arguments.

### Aliases

No aliases.

### Example

The following is an example of how to use the **ames** subcommand:

```
KDB(0)> ames
VMM AMEs
Select the ame to display by:
 1) current process
 2) specified process
 3) specified address map
Enter your choice: 2
Enter the process id: 0326E
Switch to proc: E2006400

VMM address map, address D0000000
previous entry      (vme_prev)        : D0000040
next entry          (vme_next)        : D0000040
start of range      (min_offset)      : 30000000
end of range        (max_offset)      : F0000000
number of entries   (nentries)        : 00000001
size                (size)            : 00100000
non-directed map.   (min_offset2)     : 30000000
reference count     (ref_count)       : 00000001
hint                (hint)            : D0000040
first free hint     (first_free)      : D0000040
entries pageable    (entries_pageable): 00000000

VMM map entry, address D0000040
previous entry      (vme_prev)      : D0000000
next entry          (vme_next)      : D0000000
start address       (vme_start)     : 30000000
end address         (vme_end)       : 30100000
object (vnode ptr)  (object)        : 14F1B380
page num in object  (obj_pno)       : 00000000cur protection      (protection)
  : 00000003
max protection      (max_protection): 00000007
inheritance         (inheritance)   : 00000000
source sid          (source_sid)    : 0000E347
mapping sid         (mapping_sid)   : 00008344
paging sid          (paging_sid)    : 007FFFFF
original page num   (orig_obj_pno)  : 00000000
```

```
shared memory desc. (sp)            : 00000000
KDB(0)> scb 2    // display mapping sid
Enter the sid (in hex): 00008344  // sid value

VMM SCB Addr B0489BEC Index 00000344 of 0000050B  Segment ID: 00008344

  //MAPPING SEGMENT
> (_segtype)..... mapping segment
segment info bits        (_sibits)  : 10000000
default storage key      (_defkey)  : 0
starting ame             (same)     : D0000040
ending ame               (eame)     : D0000040
hint ame                 (hame)     : D0000040
waitlist for change      (msegwait) : 00000000
> (mappings).... mappings exist
sibling mmap fork seg    (sibling)  : 00000000
class ID                 (classid)  : 00000000     0
physical attachments     (_att)     : 00000000
mmap reference count     (refcnt)   : 00000001
non-fblu pageout count   (npopages) : 0000
xmem attach count        (xmemcnt)  : 0000
pages in real memory     (npages)   : 00000000
pinned pages in memory   (npinpages): 00000000
lru pageout count        (npopages) : 00000000
proc pointer             (proc)     : E2006400
(0)> more (^C to quit) ?
page frame at head       (sidlist)  : FFFFFFFF
max assigned page number (maxvpn)   : FFFFFFFF
lock                     (lock)     :@B0489C44 00000000
KDB(0)>
```

## apt subcommand

### Purpose

The **apt** subcommand provides options for display of information from the alias page table.

### Syntax

**apt** [*menu options*]

### Parameters

- *menu options* - Menu options and parameters can be entered along with the subcommand to avoid display of menus and prompts.

If this subcommand is invoked without arguments, menus and prompts are used to determine the data to be displayed. If the menu selections and required values are known, they can be entered as subcommand arguments.

### Aliases

No aliases.

### Example

The following is an example of how to use the **apt** subcommand:

```
Example:
KDB(0)> apt
VMM APT
Select the APT function:
 1) display by index
 2) display by sid,pno
 3) display by page frame
 4) count valid, free
 5) count free from pf_aptfree
 6) count valid from AHAT
 7) display free list
Enter your choice: 1
Enter the index (in hex): 0

VMM APT Entry 00000000 of 00010000
> valid
segment identifier    (sid)  : 0002A015
page number           (pno)  : 0000
page frame            (nfr)  : 00000000
protection key        (key)  : 3
storage control attr  (wimg) : 2
next on hash          (next) : FFFF
next on alias list    (anext): FFFF
next free/pin count   (free) : 0001
KDB(0)> apt 2
Enter the sid (in hex): 2a015
Enter the pno (in hex): 0

VMM APT Entry 00000000 of 00010000
> valid
segment identifier    (sid)  : 0002A015
page number           (pno)  : 0000
page frame            (nfr)  : 00000000
protection key        (key)  : 3
storage control attr  (wimg) : 2
next on hash          (next) : FFFF
next on alias list    (anext): FFFF
next free/pin count   (free) : 0001
```

```
KDB(0)> apt 4
There are  10000 APT slots allocated.
            12 are valid
          FFEE are free
KDB(0)> apt 7
000012 - 000013 - 000014 - 000015 - 000016 - 000017 - 000018 - 000019 -
00001A - 00001B - 00001C - 00001D - 00001E - 00001F - 000020 - 000021 -
000022 - 000023 - 000024 - 000025 - 000026 - 000027 - 000028 - 000029 -
00002A - 00002B - 00002C - 00002D - 00002E - 00002F - 000030 - 000031 -
000032 - 000033 - 000034 - 000035 - 000036 - 000037 - 000038 - 000039 -
00003A - 00003B - 00003C - 00003D - 00003E - 00003F - 000040 - 000041 -
000042 - 000043 - 000044 - 000045 - 000046 - 000047 - 000048 - 000049 -
00004A - 00004B - 00004C - 00004D - 00004E - 00004F - 000050 - 000051 -
000052 - 000053 - 000054 - 000055 - 000056 - 000057 - 000058 - 000059 -
00005A - 00005B - 00005C - 00005D - 00005E - 00005F - 000060 - 000061 -
000062 - 000063 - 000064 - 000065 - 000066 - 000067 - 000068 - 000069 -
00006A - 00006B - 00006C - 00006D - 00006E - 00006F - 000070 - 000071 -
000072 - 000073 - 000074 - 000075 - 000076 - 000077 - 000078 - 000079 -
00007A - 00007B - 00007C - 00007D - 00007E - 00007F - 000080 - 000081 -
000082 - 000083 - 000084 - 000085 - 000086 - 000087 - 000088 - 000089 -
00008A - 00008B - 00008C - 00008D - 00008E - 00008F - 000090 - 000091 -
000092 - 000093 - 000094 - 000095 - 000096 - 000097 - 000098 - 000099 -
00009A - 00009B - 00009C - 00009D - 00009E - 00009F - 0000A0 - 0000A1 -
0000A2 - 0000A3 - 0000A4 - 0000A5 - 0000A6 - 0000A7 - 0000A8 - 0000A9 -
0000AA - 0000AB - 0000AC - 0000AD - 0000AE - 0000AF - 0000B0 - 0000B1 -
0000B2 - 0000B3 - 0000B4 - 0000B5 - 0000B6 - 0000B7 - 0000B8 - 0000B9 -
0000BA - 0000BB - 0000BC - 0000BD - 0000BE - 0000BF - 0000C0 - 0000C1 -
(0)> more (^C to quit) ?
<snip>
```

## frameset subcommand

### Purpose

The **frameset** displays information about VMM frame sets.

### Syntax

**frameset** [*frs_id*]

### Parameters

- *frs_id* – Can be the * character to specify a summary of the frame set table should be displayed. Or, it can be a specific frameset id to indicate detailed information about the specific frameset should be displayed.

  **Note:** The **frameset** subcommand requires a parameter.

### Aliases

**frs**

### Example

The following is an example of how to use the **frameset** subcommand:

```
KDB(1)> frameset *

                 VMP MEMP FRS  NEXT_FRS  NB_PAGES  NUMFRB
memp_frs+000000  00  000  000  00000001  0013B2BC  00128CFB
memp_frs+000080  00  000  001  FFFFFFFF  0013B2BA  00128D11
KDB(1)> frameset 1

Frame Set [1] [0000000000EC7080]

> valid
freefwd              (freefwd)       : 000000000009C7D5
freebwd              (freebwd)       : 000000000009C8F3
free nfr lock @ 0000000000EC7080 00000000
free frames          (numfrb)        : 0000000000128D11
number of frames     (nb_frame)      : 000000000013B2BA
next frameset        (next_frs)      : FFFFFFFF
owning mempool       (memp_id)       : 00000000
owning vmpool        (vmpool_id)     : 00000000
KDB(1)>
```

# free subcommand

## Purpose

The **free** subcommand counts the number of free page frames.

## Syntax

**free**

## Parameters

No parameters are supported for the **free** subcommand.

The **free** subcommand counts and displays the number of free page frames, on a vmpool/frameset basis.

**Note:** The time it takes for this command to complete depends on the amount of system memory being considered. Noticeable delays are not unusual.

## Aliases

No aliases.

## Example

The following is an example of how to use the **free** subcommand:

```
KDB(1)> free

VMPOOL: 00
frame set         0  :  128CFB  free frames
frame set         1  :  128D11  free frames
KDB(1)>
```

## freelist subcommand

### Purpose

The **freelist** subcommand displays VMM free list information.

### Syntax

**freelist** [*frs_id*]

### Parameters

- *frs_id* – Specifies the frameset identifier for which you want to display VMM free list information.

The **freelist** subcommand requires an *frs_id* parameter to identify the particular frameset to examine. The list of all page frames on the free list for that frameset is then displayed.

**Note:** The longer the length of the free list, the more time this subcommand takes to complete.

### Aliases

No aliases.

### Example

The following is an example of how to use the **freelist** subcommand:

```
KDB(0)> freelist 1
00000261A5 - 00000261B5 - 00000261A3 - 00000261B1 - 00000261AF - 00000261AD -
00000261AB - 00000261A9 - 00000261A7 - 000002619B - 00000261A1 - 000002619F -
000002619D - 0000026189 - 0000026199 - 0000026197 - 0000026195 - 0000026193 -
0000026191 - 000002618F - 000002618D - 000002618B - 0000026183 - 0000026187 -
0000026185 - 0000024951 - 0000024AFD - 0000024AEB - 0000024D09 - 000002616D -
0000026121 - 0000024B9B - 0000024B9D - 000002613D - 0000024D11 - 0000024D15 -
0000024AFB - 000002617D - 0000024BC3 - 000002617B - 0000024D77 - 0000026179 -
<snip>
00000261FD - 00000261FB - 00000261F9 - 00000261F7 - 00000261F5 - 00000261F3 -
00000261F1 - 00000261EF - 00000261ED - 00000261EB - 00000261E9 - 00000261E7 -
00000261E5 - 00000261E3 - 00000261E1 - 00000261DF - 00000261DD - 00000261DB -
00000261D9 - 00000261D7 - 00000261D5 - 00000261D3 - 00000261D1 - 00000261CF -
00000261CD - 00000261CB - 00000261C9 - 00000261C7 - 00000261C5 - 00000261C3 -
00000261C1 - 00000261BF - 00000261BD - 00000261BB - 00000261B9 - 00000261B7 -
FBANCH
 2905E  free frames
KDB(0)>
```

# ipc subcommand

## Purpose

The **ipc** subcommand reports interprocess communication facility information.

## Syntax

**ipc** [*menu options*]

## Parameters

- *menu options* - Menu options and parameters can be entered along with the subcommand to avoid display of menus and prompts.

If this subcommand is invoked without parameters, then menus and prompts are used to determine the data to be displayed. If the menu selections and required values are known, you can enter them as subcommand parameters.

```
ipc 1 [1..3] to print message queue information
ipc 2 [1..2] to print shared memory information
ipc 3 [1..2] to print semaphore information
```

## Aliases

No aliases.

## Example

The following is an example of how to use the **ipc** subcommand:

```
KDB(0)> ipc
IPC info
Select the display:
 1) Message Queues
 2) Shared Memory
 3) Semaphores
Enter your choice: 1
 1) all msqid_ds
 2) select one msqid_ds
 3) struct msg
Enter your choice: 1
Message Queue ID 00000000 @ D0000000
uid.......... 48454150 gid.......... 00043000
cuid......... 00000000 cgid......... 00000001
mode......... 0000FFBD seq.......... 0000
key.......... 40000000
msg_first.... 00000000
msg_last..... 00000000
msg_cbytes... 00000000 msg_qnum..... 00000000
msg_qbytes... 00000000
msg_lspid.... 00000000
msg_lrpid.... 00000000
msg_stime.... 00000000
msg_rtime.... 00000000
msg_ctime.... 00000000
msg_rwait.... 00000000 msg_wwait.... 00000000
msg_reqevents. 0000
msg_next..... 00000000
msg_prev..... 00000000
orig_msqid... 00000000 cur_msqid.... 00000000 crid......... 00000000
vhat_next.... 00000000
vhat_prev.... 00000000
rt_ipcx...... 00000000
```

```
maxmsg........ 00000000
notify........ NULL
KDB(0)>
```

## rtipc subcommand

### Purpose

The **rtipc** subcommand reports posix realtime interprocess communication facility information.

### Syntax

**rtipc** [*menu options*]

### Parameters

- *menu options* – Identifies menu options and parameters that can be entered along with the subcommand to avoid display of menus and prompts.

If this subcommand is invoked without parameters, then menus and prompts are used to determine the data to be displayed.

```
(0)> rtipc
RTIPC info
Select the display:
  1) Message Queues
  2) Shared Memory
  3) Semaphores
  4) Message Queue Name Table
  5) Shared Memory Name Table
  6) Semaphore Name Table
 Enter your choice:
```

Reported information is related to posix realtime message queues, shared memory and semaphores, and their associated name table.

**Note:** If the menu selections and required values are known, they can be entered as subcommand parameters.

For realtime ipc objects, displayed data can be selected by object address, index in object table, or realtime ipc name. If selection is by name, the subcommand must be invoked with all its parameters.

```
(0)> rtipc 1
  1) all entries
  2) select one entry by address
  3) select one entry by index
  4) select one entry by name
     (name up to 16 chars, type command in once)
 Enter your choice:
```

For a realtime ipc name table, displayed data can be selected by index in the name table.

```
(0)> rtipc 4
  1) all entries
  2) select one entry by index
 Enter your choice:
```

### Aliases

No aliases.

### Example

The following is an example of how to use the **rtipc** subcommand:

```
(0)> rtipc
RTIPC info
Select the display:
 1) Message Queues
```

```
 2) Shared Memory
 3) Semaphores
 4) Message Queue Name Table
 5) Shared Memory Name Table
 6) Semaphore Name Table
Enter your choice: 1
 1) all entries
 2) select one entry by address
 3) select one entry by index
 4) select one entry by name
    (name up to 16 chars, type command in once)
Enter your choice: 2
Enter the address (in hex): F10000B08013BD98

RT Message Queue  idx 00007E57  @ F10000B08013BD98
next.......... 0000000000000000
name.......... mymq
sysVid........ 000C7E59
flags......... 00000001 INUSE
refcnt........ 00000000
msgsize....... 00000400

(0)> rtipc 4 1
00000030 : F10000B080360998
00000061 : F10000B08026A520
00000062 : F10000B08029C458 F10000B08025B520
00000064 : F10000B080267B18 F10000B080279368 F10000B08026F430
0000006A : F10000B080269F80
```

## rtipcd subcommand

### Purpose

The **rtipcd** subcommand reports posix realtime ipc descriptor information.

### Syntax

**rtipcd** [*menu options*]

### Parameters

- *menu options* – Use menu options and parameters with the subcommand to avoid display of menus and prompts.

If this subcommand is invoked without parameters, then menus and prompts are used to determine which data to display.

Reported information is related to process descriptors of posix realtime message queues and semaphores, and process descriptor hash tables.

```
0)> rtipcd
RTIPC Descriptor info
Select the display:
 1) Message Queue Descriptors
 2) Semaphore Descriptors
 3) Message Queue Descriptor Table
 4) Semaphore Descriptor Table
Enter your choice:
```

For realtime ipc descriptors, displayed data can be selected by descriptor address or descriptor user id.

```
0)> rtipcd 1
 1) select one entry by address
 2) select one entry by user id
Enter your choice:
```

For realtime ipc descriptor tables, displayed data can be selected by hash table index.

```
(0)> rtipcd 3
 1) all entries
 2) select one entry by index
Enter your choice:
```

### Aliases

No aliases.

### Example

The following is an example of how to use the **rtipcd** subcommand:

```
(0)> rtipcd
RTIPC Descriptor info
Select the display:
 1) Message Queue Descriptors
 2) Semaphore Descriptors
 3) Message Queue Descriptor Table
 4) Semaphore Descriptor Table
Enter your choice: 1
 1) select one entry by address
 2) select one entry by user id
Enter your choice: 1
Enter the address (in hex): F100009E189B5C00
```

```
RT Message Queue Descriptor  @ F100009E189B5C00
next.......... F100009E189B5F00
rt_ipcx....... 0001AD34
mq oflags..... 00000003 READ WRITE
mq umqid...... 68000000  idx.. 0034 seq.. 00000000


(0)> rtipcd 3 1
0000001C : F100009E189B57E0
00000034 : F100009E189B5C00 F100009E189B5F00
00000037 : F100009E189B5AE0
```

## lka subcommand

### Purpose

The **lka** subcommand displays VMM lock anchor data and data for the transaction blocks in the transaction block table. You can display individual entries of the transaction block table by providing a slot number or an effective address.

### Syntax

**lka** [*slot* | *effectiveaddress*]

### Parameters

- *slot* – Specifies the slot number in the transaction block table to be displayed. This parameter must be a decimal value.
- *effectiveaddress* – Specifies the effective address of an entry in the transaction block table. Symbols, hexadecimal values, or hexadecimal expressions can be used to specify the address.

### Aliases

**lockanch**, **tblk**

### Example

The following is an example of how to use the **lka** subcommand:

```
KDB(0)> lka

VMM LOCKANCH lkwseg  +000000

nexttid...... : 0000B210 freetid...... : 00000002 maxtid....... : 00000002
lwptr........ : D000B000 freelock..... : 00000006 morelocks.... : D000C000
syncwait..... : 00000000 tblkwait..... : 00000000 freewait..... : 00000000
lw_lock...... @ 006F08E0 00000000
tblk......... @ D0000024 lockhash..... @ D000A024
  @tblk[0] lkwseg  +000024
logtid.... 00000000 next...... 00000000 tid....... 00000000 flag...... 00000000
cpn....... 00000000 ceor...... 00000000 cxor...... 00000000 csn....... 00000000
waitsid... 00000000 waitline.. 00000000 locker.... 00000000 lsidx..... 00000000
gcw.elist. 00000000 gcw.owner. 00000000 gcw.lock.. 00000000 gcw.boost. 00000000
logage.... 00000000 waitors... 00000000 cqnext.... 00000000
  @tblk[1] lkwseg  +000074  tblk[1].cqnext lkwseg  +0000C4
logtid.... 0000A72A next...... 00000003 tid....... 00000001 flag...... 0000002D
cpn....... 00001AC6 ceor...... 00000530 cxor...... 1D696F24 csn....... 00000003
waitsid... 00000000 waitline.. 00000000 locker.... 00000000 lsidx..... 0000008F
gcw.elist. FFFFFFFF gcw.owner. 00000000 gcw.lock.. 00000000 gcw.boost. 00000000
logage.... 00000000 waitors... 00000000 cqnext.... D00000C4
flag...... QUEUE COMMIT COMMITTED LEADER
  @tblk[2] lkwseg  +0000C4
(0)> more (^C to quit) ?
logtid.... 0000B210 next...... 00000001 tid....... 00000002 flag...... 00000000
cpn....... 00000000 ceor...... 00000000 cxor...... 00000000 csn....... 00000000
waitsid... 00000000 waitline.. 00000000 locker.... 00000000 lsidx..... 0000008F
gcw.elist. FFFFFFFF gcw.owner. 00000000 gcw.lock.. 00000000 gcw.boost. 00000000
logage.... 00000000 waitors... 00000000 cqnext.... 00000000
KDB(0)>
```

## lkh subcommand

### Purpose

The **lkh** subcommand displays the contents of the VMM lock hash list. The entries for a particular hash chain can be viewed by specifying the slot number or effective address of an entry in the VMM lock hash list.

### Syntax

**lkh** [*slot* | *eaddr*]

### Parameters

- *slot* - Specifies the slot number in the VMM lock hash list. This parameter must be a decimal value.
- *eaddr* - Specifies the effective address of a VMM lock hash list entry. Symbols, hexadecimal values, or hexadecimal expressions can be used in specification of the address.

### Aliases

**lockhash**

### Example

The following is an example of how to use the **lkh** subcommand:

```
KDB(0)> lkh
                BUCKET HEAD     COUNT

lkwseg  +00F090   22   00000001     1
KDB(0)> lkh @r3
HASH ENTRY(  1): F100009C0000F03C
KDB(0)> dr r3
r3  : 0000000000000001   00000001
KDB(0)> lkh 1
HASH ENTRY(  1): F100009C0000F03C
```

## lkw subcommand

## Purpose

The **lkw** subcommand displays VMM lock words.

## Syntax

**lkw** [*slot* | *effectiveaddress*]

## Parameters

- *slot* – Specifies the slot number of an entry in the VMM lock word table. This parameter must be a decimal value.
- *effectiveaddress* – Specifies the effective address of an entry in the VMM lock word table. Symbols, hexadecimal values, or hexadecimal expressions can be used to specify the address.

If no parameter is entered, a summary of the entries in the VMM lock word table is displayed, one line per entry. If a parameter identifying a particular entry is entered, details are shown for that entry and the following entries on the transaction ID chain.

## Aliases

**lockword**

## Example

The following is an example of how to use the **lkw** subcommand:

```
KDB(0)> lkw
                         NEXT   TIDNXT      SID PAGE  TID FLAGS

   0 lkwseg  +00B000       0       0 00000000 0000 0000
   1 lkwseg  +00B028       4       3 000300D8 0002 0002 WRITE FREE
   2 lkwseg  +00B050       3       6 00028074 0001 0002 WRITE FREE
   3 lkwseg  +00B078       1       2 00028074 0000 0002 WRITE FREE
   4 lkwseg  +00B0A0       5       1 000300D8 0008 0001 WRITE FREE
   5 lkwseg  +00B0C8       7       4 000300D8 0003 0001 WRITE FREE
   6 lkwseg  +00B0F0       2       0 000100A8 018F 0002 WRITE FREE
   7 lkwseg  +00B118       8       0 00000000 0000 0000
   8 lkwseg  +00B140       9       0 00000000 0000 0000
   9 lkwseg  +00B168      10       0 00000000 0000 0000
  10 lkwseg  +00B190      11       0 00000000 0000 0000
  11 lkwseg  +00B1B8      12       0 00000000 0000 0000
  12 lkwseg  +00B1E0      13       0 00000000 0000 0000
  13 lkwseg  +00B208      14       0 00000000 0000 0000
  14 lkwseg  +00B230      15       0 00000000 0000 0000
  15 lkwseg  +00B258      16       0 00000000 0000 0000
  16 lkwseg  +00B280      17       0 00000000 0000 0000
  17 lkwseg  +00B2A8      18       0 00000000 0000 0000
  18 lkwseg  +00B2D0      19       0 00000000 0000 0000
  19 lkwseg  +00B2F8      20       0 00000000 0000 0000
  20 lkwseg  +00B320      21       0 00000000 0000 0000
  21 lkwseg  +00B348      22       0 00000000 0000 0000
  22 lkwseg  +00B370      23       0 00000000 0000 0000
  23 lkwseg  +00B398      24       0 00000000 0000 0000
  24 lkwseg  +00B3C0      25       0 00000000 0000 0000
  25 lkwseg  +00B3E8      26       0 00000000 0000 0000
  26 lkwseg  +00B410      27       0 00000000 0000 0000
  27 lkwseg  +00B438      28       0 00000000 0000 0000
  28 lkwseg  +00B460      29       0 00000000 0000 0000
  <snip>
KDB(0)> lkw 3
                         NEXT   TIDNXT      SID PAGE  TID FLAGS
```

```
   3 lkwseg  +00B078        1        2 00028074 0000 0002 WRITE FREE
bits........... 20000000 log............ 01B41588
home........... 00000020 extmem......... 00000000
                         NEXT   TIDNXT      SID PAGE  TID FLAGS
   2 lkwseg  +00B050        3        6 00028074 0001 0002 WRITE FREE
bits........... 10000000 log............ 01B41588
home........... 00000021 extmem......... 00000000
                         NEXT   TIDNXT      SID PAGE  TID FLAGS
   6 lkwseg  +00B0F0        2        0 000100A8 018F 0002 WRITE FREE
bits........... 00020000 log............ 01B51C88
home........... 0000300F extmem......... 00000000
KDB(0)>
```

## mempool subcommand

### Purpose

The **mempool** subcommand displays information about VMM memory pools.

### Syntax

**mempool** [*memp_id*]

### Parameters

- *memp_id* – Is the asterisk (*****) character or a memory pool identifier. The asterisk (*****) displays a summary of the memory pool table. A specific memory pool identifier displays detailed information about the specific memory pool.

  **Note:** The **mempool** subcommand requires a parameter.

### Aliases

**memp**

### Example

The following is an example of how to use the **mempool** subcommand:

```
KDB(1)> mempool *
                 VMP MEMP  NB_PAGES  FRAMESETS    NUMFRB
memp_frs+040000  00  000   00276576  000 001      00251A0C
KDB(1)> mempool 0

Memory Pool [0] [00F07000]
Frame Sets:
        [00000000] [00EC7000]
        [00000001] [00EC7080]

> valid
number of frames     (nb_frame)          : 0000000000276576
first frame set      (first_frs)         : 00000000
next memory pool     (next)              : FFFFFFFF
owning vmpool        (vmpool_id)         : 00000000

LRU statistics and thresholds

min perm frames      (minperm)           : 000000000007AA86
max perm frames      (maxperm)           : 00000000001EAA18
max client frames    (maxclient)         : 00000000001EAA18
fblru page-outs      (numpout)           : 0000000000000000
fblru remote pg-outs (numremote)         : 0000000000000000
num client frames    (numclient)         : 0000000000000000
compressed segs      (numcompress)       : 0000000000000000
num perm frames      (numperm)           : 0000000000001940
(1)> more (^C to quit) ?
comp repage cnt      (rpgcnt[RPCOMP])    : 0000000000000000
file repage cnt      (rpgcnt[RPFILE])    : 0000000000000000
freewake             (freewake)          : 00000000
free frame wait      (freewait)          : 0000000000000000
v_sync cursor        (syncptr)           : 00000000
next lru candidate   (lruptr)            : 0000000000000D21
frames examined      (lrucnt)            : 0000000000000000
start of bucket      (lrumin)            : 0000000000000D22
end of bucket        (lrumax)            : FFFFFFFFFFFFFFFF
LRU bucket size      (lrubucket)         : 0000000000020000
lru interval head    (lrumem)            : F100001420000080
nolru interval head  (nolru)             : F1000014200000C0
```

```
index in int array   (lruidx)           : F100001420000300
lru index for bucket (saveidx)          : F100001420000300
force fileonly off   (fileonly_off)     : 00000000
lru daemon anchor    (lru_daemon)       : F100009E14741400
lru request          (lru_requested)    : 00000000
DR thread id         (dr_tid)           : FFFFFFFF
lru pf color         (lru_pf_color)     :        0
LRU      lock @ 0000000000F07008: 00000000
KDB(1)>
```

## pdt subcommand

### Purpose

The **pdt** subcommand displays entries of the paging device table.

### Syntax

**pdt** [*] [*slot*]

### Parameters

- * – Displays all entries of the paging device table.
- *slot* – Specifies the slot number within the paging device table to be displayed. This value must be a hexadecimal value.

An asterisk ( * ) parameter displays all entries in a summary. To display the details for a specific entry, specify the slot number in the paging device table. If no parameter is specified, you are prompted to enter the PDT index you want to display. Detailed data is then displayed for the entered slot and all higher slot numbers.

### Aliases

No aliases.

### Example

The following is an example of how to use the **pdt** subcommand:

```
KDB(0)> pdt *                   // display paging device table

               SLOT   NEXTIO   DEVICE   IOTAIL  DMSRVAL    IOCNT   <name>
vmmdseg+460000 0000 FFFFFFFF 000A0002 FFFFFFFF 00000000 00000000 paging
vmmdseg+460580 0010 FFFFFFFF 06067A2C FFFFFFFF 00000000 00000000 remote
vmmdseg+4605D8 0011 FFFFFFFF 000A0007 FFFFFFFF 00002081 00000000 filesystem
vmmdseg+460630 0012 FFFFFFFF 000A0003 FFFFFFFF 00000000 00000000 log
vmmdseg+460688 0013 FFFFFFFF 000A0004 FFFFFFFF 0003609B 00000000 filesystem
vmmdseg+4606E0 0014 FFFFFFFF 000A0005 FFFFFFFF 000140AA 00000000 filesystem
vmmdseg+460738 0015 FFFFFFFF 000A0006 FFFFFFFF 000340DA 00000000 filesystem
vmmdseg+460790 0016 FFFFFFFF 06067A8C FFFFFFFF 00000000 00000000 remote
vmmdseg+4607E8 0017 FFFFFFFF 000A0008 FFFFFFFF 0001422A 00000000 filesystem
vmmdseg+460840 0018 FFFFFFFF 000A0009 FFFFFFFF 00020230 00000000 filesystem
vmmdseg+460898 0019 FFFFFFFF 000A000B FFFFFFFF 00000000 00000000 local client
vmmdseg+4608F0 001A FFFFFFFF 0222D694 FFFFFFFF 00000000 00000000 remote
KDB(0)> pdt 13   // display paging device table slot 13

PDT address B0460688 entry 0013 of 03FF, type: FILESYSTEM
next pdt on i/o list  (nextio)  : FFFFFFFF
dev_t or strategy ptr (device)  : 000A0004
last frame w/pend I/O (iotail)  : FFFFFFFF
free buf_struct list  (bufstr)  : 300861B8
total buf structs     (nbufs)   : 00BA
available (PAGING)    (avail)   : 0000
JFS  disk agsize      (agsize)  : 0800
JFS inode agsize      (iagsize) : 1000
JFS log SCB index     (logsidx) : 0008F
JFS fragments per page(fperpage): 1
JFS compression type  (comptype): 0
JFS log2 bigalloc mult(bigexp)  : 0
disk map srval        (dmsrval) : 0003609B
i/o's not finished    (iocnt)   : 00000000
device wait list (devwait)      : 00000000
buffer wait list (bufwait)      : 00000000
logical volume lock (lock)      :@B04606B8 00000000
```

```
buffer list lock    (buf_lock)  :@B04606BC 00000000
flag bits            (devflags) : 80000000
max phys Xlation ent (maxphys)  : 00000020
SR val for .indirect (indsrval) : 00030098
SR val for .inodes   (inosrval) : 00032099
SR val for .inodemap (imsrval)  : 0003409A
KDB(0)>
```

# pfhdata subcommand

## Purpose

The **pfhdata** subcommand displays virtual memory control variables.

## Syntax

**pfhdata**

## Parameters

No parameters.

## Aliases

No aliases.

## Example

The following is an example of how to use the **pfhdata** subcommand:

```
KDB(0)> pfhdata

VMM Control Variables: B0476000 vmmdseg +476000

1st free sid entry   (sidfree)          : 000004A5
1st delete pending   (sidxmem)          : 00000000
highest sid entry    (hisid)            : 0000050C
frames not pinned    (pfavail)          : 0005965F
app frames free      (pfpinavail)       : 000511B6
lru bucket size      (lrubucket)        : 00020000
last pdt on i/o list (iotail)           : FFFFFFFF
num of paging spaces (npgspaces)        : 00000001
PDT last alloc from  (pdtlast)          : 00000000
max pgsp PDT index   (pdtmaxpg)         : 00000000
PDT free pool list   (pdtfree)          : 00000000
PDT high watermark   (pdtmax)           : 0000001B
PDT index of server  (pdtserver)        : 00000000
scb serial num       (nxtscbnum)        : 0000060D
num of comp replaces (nreplaced[RPCOMP]): 00000000
num of file replaces (nreplaced[RPFILE]): 00000000
num of comp repages  (nrepaged[RPCOMP]) : 00000000
num of file repages  (nrepaged[RPFILE]) : 00000000
min page-ahead       (minpgahead)       : 00000002
max page-ahead       (maxpgahead)       : 00000008
sysbr protect key    (kerkey)           : 00000000
non-ws page-outs     (numpermio)        : 00000000
free frame wait      (freewait)         : 00000000
device i/o wait      (devwait)          : 00000000
extend XPT wait      (extendwait)       : 00000000
buf struct wait      (bufwait)          : 00000000
inh/delete wait      (deletewait)       : 00000000
SIGDANGER level      (npswarn)          : 00001000
SIGKILL level        (npskill)          : 00000400
next warn level      (nextwarn)         : 00001000
next kill level      (nextkill)         : 00000400
adj warn level       (adjwarn)          : 00000008
adj kill level       (adjkill)          : 00000008
cur pdt alloc        (npdtblks)         : 00000002
max pdt alloc        (maxpdtblks)       : 00000004
num i/o sched        (numsched)         : 00000004
disk quota wait      (dqwait)           : 00000000
1st free ame entry   (amefree)          : 0000000A
1st del pending ame  (amexmem)          : 00000000
highest ame entry    (hiame)            : 00000040
```

```
pag space free wait  (pgspwait)         : 00000000
first free apt entry (aptfree)          : 00000012
apt high water mark  (hiapt)            : 0000FFFF
next apt entry       (aptlru)           : 00000000
first free esid      (esidfree)         : 00200054
high index of esid   (hiesid)           : 00000160
first lgpg rsvd sidx (sidxlimit)        : 00200000
log high wartermark  (logmax)           : 00000002
sid index of logs    (logsidx)          : B0476734
lru creation thread  (lruwait)          : 00000000
memp needing daemon  (mempnew)          : 00000000
minperm percent      (minperm)          : 20.0 %
maxperm percent      (maxperm)          : 80.0 %
maxclient percent    (maxclient)        : 80.0 %
frame thresholds     (minfree, maxfree)
computational                          : 00000078 00000080
client                                 : 00000078 00000080
persistent                             : 00000078 00000080
fixed lmb freelist   (fixlmbfree)       : 00000001
fixed lmb size(pages)(fixlmbsz)         : 00000000
fixed lmb firstnfr   (fixlmbfirst)      : 00000001
fixed lmb lastnfr    (fixlmblast)       : 00000001
vmpool being deleted   (vmp_del)        : 00000000
mempool being deleted  (memp_del)       : 00000000
frameset being deleted (frs_del)        : 00000000
global vmap      lock @ B0476100 00000000
global ame       lock @ B0476180 00000000
global rpt       lock @ B0476200 00000000
rpt pool    lock [00] @ B0476280 00000000
rpt pool    lock [01] @ B0476284 00000000
rpt pool    lock [02] @ B0476288 00000000
rpt pool    lock [03] @ B047628C 00000000
rpt pool    lock [04] @ B0476290 00000000
rpt pool    lock [05] @ B0476294 00000000
rpt pool    lock [06] @ B0476298 00000000
rpt pool    lock [07] @ B047629C 00000000
rpt pool    lock [08] @ B04762A0 00000000
rpt pool    lock [09] @ B04762A4 00000000
rpt pool    lock [10] @ B04762A8 00000000
rpt pool    lock [11] @ B04762AC 00000000
rpt pool    lock [12] @ B04762B0 00000000
rpt pool    lock [13] @ B04762B4 00000000
rpt pool    lock [14] @ B04762B8 00000000
rpt pool    lock [15] @ B04762BC 00000000
global alloc     lock @ B0476300 00000000
apt freelist     lock @ B0476380 00000000
pdt allocation   lock @ B0476400 00000000
pdt io list      lock @ B0476480 00000000
compression page lock @ B0476500 00000000
serv frame alloc lock @ B0476580 00000000
fixlmb freelist  lock @ B0476600 00000000
KDB(0)>
```

# pft subcommand

## Purpose

The **pft** subcommand displays information about the VMM page frame table.

## Syntax

**pft** [*menu options*]

## Parameters

- *menu options* – Use menu options and parameters with the subcommand to avoid display of menus and prompts.

If the **pft** subcommand is invoked without parameters, then menus and prompts determine which data is displayed. If the menu selections and required values are known, you can enter them as subcommand parameters.

## Aliases

No aliases.

## Example

The following is an example of how to use the **pft** subcommand:

```
KDB(0)> pft
VMM PFT
Select the PFT entry to display by:
 1) page frame #
 2) h/w hash (sid,pno)
 3) s/w hash (sid,pno)
 4) search on swbits
 5) search on pincount
 6) search for hidden pages
 7) scb list
 8) io list
 9) deferred pgsp service frames
 a) scb list (compact output)
 b) ksp list (compact output)
Enter your choice: 1
Enter the page frame number (in hex): FCD

VMM PFT Entry For Page Frame 0000000FCD of 000005FFFF

pte = 00000000095F9700 pvt = 0000000000C03F34 pft = 000000000203B40C
h/w hashed sid : 0000000024012  pno : 000000FF3C  key : 0
source     sid :      00024012  pno : 000000FF3C  key : 0

> in use
> on scb list
> valid (h/w)
> referenced (pft/pvt/pte): 0/0/1
> modified (pft/pvt/pte): 1/0/1
owning vmpool id      (vmp)    : 0000
owning mempool id     (memp)   : 0000
owning frameset       (frs)    : 0001
page number in scb    (spage)  : FF3C
disk block number     (dblock) : 00000000
next page on scb list (sidfwd) : 00000FC6
prev page on scb list (sidbwd) : 0005F6D4
freefwd/waitlist      (freefwd): 00000000
freebwd/logage/pincnt (freebwd): 00010000
out-of-order I/O      (nonfifo): 00000000
```

```
(0)> more (^C to quit) ?
storage attributes    (wimg)   : 2
next page on s/w hash (next)   : FFFFFFFF
List of alias entries (alist)  : 0000FFFF
index in PDT          (devid)  : 0000
next frame i/o list   (nextio) : 00000000
save key across pagein(savekey): 0
KDB(0)> pft 2
Enter the sid (in hex): 24012
Enter the pno (in hex): FF3C

VMM PFT Entry For Page Frame 0000000FCD of 000005FFFF

pte = 00000000095F9700 pvt = 0000000000C03F34 pft = 000000000203B40C
h/w hashed sid : 0000000024012  pno : 000000FF3C  key : 0
source     sid :      00024012  pno : 000000FF3C  key : 0

> in use
> on scb list
> valid (h/w)
> referenced (pft/pvt/pte): 0/0/1
> modified (pft/pvt/pte): 1/0/1
owning vmpool id      (vmp)    : 0000
owning mempool id     (memp)   : 0000
owning frameset       (frs)    : 0001
page number in scb    (spage)  : FF3C
disk block number     (dblock) : 00000000
next page on scb list (sidfwd) : 00000FC6
prev page on scb list (sidbwd) : 0005F6D4
freefwd/waitlist      (freefwd): 00000000
freebwd/logage/pincnt (freebwd): 00010000
out-of-order I/O      (nonfifo): 00000000
(0)> more (^C to quit) ?
storage attributes    (wimg)   : 2
next page on s/w hash (next)   : FFFFFFFF
List of alias entries (alist)  : 0000FFFF
index in PDT          (devid)  : 0000
next frame i/o list   (nextio) : 00000000
save key across pagein(savekey): 0
KDB(0)> pft 3 24012 FF3C

VMM PFT Entry For Page Frame 0000000FCD of 000005FFFF

pte = 00000000095F9700 pvt = 0000000000C03F34 pft = 000000000203B40C
h/w hashed sid : 0000000024012  pno : 000000FF3C  key : 0
source     sid :      00024012  pno : 000000FF3C  key : 0

> in use
> on scb list
> valid (h/w)
> referenced (pft/pvt/pte): 0/0/1
> modified (pft/pvt/pte): 1/0/1
owning vmpool id      (vmp)    : 0000
owning mempool id     (memp)   : 0000
owning frameset       (frs)    : 0001
page number in scb    (spage)  : FF3C
disk block number     (dblock) : 00000000
next page on scb list (sidfwd) : 00000FC6
prev page on scb list (sidbwd) : 0005F6D4
freefwd/waitlist      (freefwd): 00000000
freebwd/logage/pincnt (freebwd): 00010000
out-of-order I/O      (nonfifo): 00000000
(0)> more (^C to quit) ?
storage attributes    (wimg)   : 2
next page on s/w hash (next)   : FFFFFFFF
List of alias entries (alist)  : 0000FFFF
index in PDT          (devid)  : 0000
```

```
next frame i/o list   (nextio) : 00000000
save key across pagein(savekey): 0
KDB(0)> pft 7
Enter the sid (in hex): 00024012

VMM PFT Entry For Page Frame 0000000FCF of 000005FFFF

pte = 00000000095FB700 pvt = 0000000000C03F3C pft = 000000000203B484
h/w hashed sid : 0000000024012  pno : 000000FF7C  key : 0
source     sid :      00024012  pno : 000000FF7C  key : 0

> in use
> on scb list
> valid (h/w)
> referenced (pft/pvt/pte): 0/0/1
> modified (pft/pvt/pte): 1/0/1
owning vmpool id      (vmp)    : 0000
owning mempool id     (memp)   : 0000
owning frameset       (frs)    : 0001
page number in scb    (spage)  : FF7C
disk block number     (dblock) : 00000000
next page on scb list (sidfwd) : 0005F6D4
prev page on scb list (sidbwd) : FFFFFFFF
freefwd/waitlist      (freefwd): 00000000
freebwd/logage/pincnt (freebwd): 00000000
out-of-order I/O      (nonfifo): 00000000
(0)> more (^C to quit) ?
storage attributes    (wimg)   : 2
next page on s/w hash (next)   : FFFFFFFF
List of alias entries (alist)  : 0000FFFF
index in PDT          (devid)  : 0000
next frame i/o list   (nextio) : 00000000
save key across pagein(savekey): 0

VMM PFT Entry For Page Frame 000005F6D4 of 000005FFFF

pte = 00000000095F9400 pvt = 0000000000D7DB50 pft = 000000000365D9B0
h/w hashed sid : 0000000024012  pno : 000000FF3A  key : 0
source     sid :      00024012  pno : 000000FF3A  key : 0

> in use
> on scb list
> valid (h/w)
> referenced (pft/pvt/pte): 0/0/1
> modified (pft/pvt/pte): 1/0/0
owning vmpool id      (vmp)    : 0000
owning mempool id     (memp)   : 0000
owning frameset       (frs)    : 0000
page number in scb    (spage)  : FF3A
(0)> more (^C to quit) ?
disk block number     (dblock) : 00000000
next page on scb list (sidfwd) : 00000FCD
prev page on scb list (sidbwd) : 00000FCF
freefwd/waitlist      (freefwd): 00000000
freebwd/logage/pincnt (freebwd): 00010000
out-of-order I/O      (nonfifo): 00000000
storage attributes    (wimg)   : 2
next page on s/w hash (next)   : FFFFFFFF
List of alias entries (alist)  : 0000FFFF
index in PDT          (devid)  : 0000
next frame i/o list   (nextio) : 00000000
save key across pagein(savekey): 0

VMM PFT Entry For Page Frame 0000000FCD of 000005FFFF

pte = 00000000095F9700 pvt = 0000000000C03F34 pft = 000000000203B40C
h/w hashed sid : 0000000024012  pno : 000000FF3C  key : 0
```

```
source     sid :      00024012  pno : 000000FF3C  key : 0

> in use
> on scb list
> valid (h/w)
(0)> more (^C to quit) ?
<snip>
KDB(0)> pft a
Enter the sid (in hex): 00024012
Frame    Ord..page Pincount Dblock Key ...
00000FCF      FF7C 00000000 00000000 K MOD REF
0005F6D4      FF3A 00010000 00000000 K MOD REF
00000FCD      FF3C 00010000 00000000 K MOD REF
00000FC6      FF3B 00020000 00000000 K MOD REF

Pages on SCB list
npages.......... 00000004
on sidlist...... 00000004
file pageout.... 00000000
pageout_pagein.. 00000000
KDB(0)> pft 5

Page frames with pincount > 0:
00000, 00002-005A3, 006F0-006F4, 0082D-00BFF, 00C0E-00C10
00C20-00C27, 00D80-00DD7, 00DDB, 00FB4, 00FB6-00FB8, 00FBB-00FC7
00FCA-00FCE, 00FD0-00FD2, 00FD4-00FD9, 00FDB, 00FDD, 00FE0-00FFF
01007, 01017, 01019, 0102C, 01033, 01038
0103A, 0103C, 0103E, 01040, 01042-01044, 01046
01048, 0104F, 01051, 01053, 01055, 01057
01059, 0105B, 0105D, 0105F, 01065, 010B4
010B6, 010B8, 010BA, 010BC, 010BE, 010C0
010C2, 010C4, 010CC, 010CE-010D1, 010D3, 010D5
010D7, 010D9, 010DB, 010DD, 010DF, 010E3
010E9, 010EB, 010ED, 010EF, 010F1, 01160
0116A, 0116C, 0116E, 01170, 01172, 01174
01176, 01178, 0117A, 0117C, 0117E, 01180
01182-01184, 01186, 01188, 0118A, 0118C, 0118E
01190, 01192, 01194, 01196-01337, 01339, 0133B
0133D, 0133F, 01341, 01343, 01345, 01347
01349, 0134B, 0134D, 0134F, 01351, 01353
01355, 01357, 01359, 0135B, 0135D, 0135F
01361, 01363-01364, 01366, 01368, 0136A, 0136C
0136E, 01370, 01372, 01374, 01376, 01378-0137A
<snip>
```

# swhat subcommand

## Purpose

The **swhat** subcommand displays VMM SW hash table entries. It can also be used to look for corrupted SW hash table entries.

## Syntax

**swhat** [1..3]

**swhat** 1 [*index*]

**swhat** 2 [*sid pno*]

**swhat** 3

## Parameters

- *index* – Indicates the **swhat** index.
- *sid* – Indicates the virtual segment identifier.
- *pno* – Indicates the page number.

When the **swhat** subcommand is given no parameters, a menu is displayed with the following options:

- 1 – Displays the software hash table entry identified by a **swhat** index entered by the user.
- 2 – Displays the software hash table entry identified by a *sid* (virtual segment identifier) and *pno* (page number) entered by the user.
- 3 – Checks for corruption in the **swhat** by examining the stored page frame numbers.

The command completes after it runs one of the options. To exit the menu and terminate the command without running any of the options, enter a period (.).

**Note:** You can enter multiple parameters simultaneously.

## Aliases

No aliases.

## Example

The following is an example of how to use the **swhat** subcommand:

```
KDB(0)> swhat
VMM SWHAT
Select the SWHAT option:
 1) display by index
 2) hash by (sid,pno)
 3) look for invalid entries
Enter your choice: 1
Enter the swhat index (in hex): 88
vmmswhat+000220 swhat[00000088]: 00000088
KDB(0)> swhat 1 88
vmmswhat+000220 swhat[00000088]: 00000088
KDB(0)> swhat 2
Enter the sid (in hex): 0
Enter the pno (in hex): 88
vmmswhat+000220 swhat[00000088]: 00000088
```

```
KDB(0)> swhat 3
There are 00000000 corrupt entries.
KDB(0)>
```

# pvt subcommand

## Purpose

The **pvt** subcommand displays the VMM PVT and PVLIST entries. The **pvt** subcommand can also be used to look for corrupted PVT and PVLIST entries.

## Syntax

**pvt** [1..4]

**pvt** 1 [*index*]

**pvt** 2

**pvt** 3 [*index*]

**pvt** 4

## Parameters

- *index* – Identifies the PVT or PVLIST index for which you want PVT information.

If you use the **pvt** subcommand with no parameters, a menu with four options is displayed. Choose one of the options, or type the parameters with the options as part of the subcommand. The options you can choose or type are the following:

- 1 – Displays the PVT identified by a PVT index entered by the user.
- 2 – Checks the PVT entry for every page with a valid software pft entry by examining the pte index stored in the PVT. Entries identified as corrupted are printed.
- 3 – Displays the PVLIST identified by a PVLIST index entered by the user.
- 4 – Checks the PVLIST entry for each pte index by examining the **pvnext** field in the PVLIST.

The subcommand terminates after running one of the options.

To exit the menu and terminate the subcommand without running any of the options, enter a period (.).

**Note:** Multiple parameters can be entered simultaneously.

## Aliases

**pvlist**

## Example

The following is an example of how to use the **pvt** subcommand:

```
KDB(0)> pvt
VMM PVT/PVLIST
Select the PVT/PVLIST option:
 1) display pvt by index
 2) look for invalid pvt entries
 3) display pvlist by index
 4) look for invalid pvlist entries
Enter your choice: 1
Enter the pvt index (in hex): 88
               NFR         PTEX      REF MOD  RAW_BITS
p64pvt+000220  0000000088  00000440  0   0    00000440
KDB(0)> pvt 1 88
               NFR         PTEX      REF MOD  RAW_BITS
p64pvt+000220  0000000088  00000440  0   0    00000440
```

```
KDB(0)> pvt 3
Enter the pvlist index (in hex): 440
INDEX     PNO  NEXT      RAW_BITS
00000440  088  3FFFFFFF  000000883FFFFFFF
KDB(0)> pvt 2
There are 00000000 corrupt entries.
KDB(0)> pvt 4
There are 00000000 corrupt entries.
KDB(0)>
```

# pta subcommand

## Purpose

The **pta** subcommand displays data from the VMM PTA segment.

## Syntax

**pta** [**-r**] [**-d**] [**-a**] [**-v**] [**-x**] [**-f** *sid* | *idx* ]

## Parameters

- **-r** – Displays XPT root data.
- **-d** – Displays XPT direct block data.
- **-a** – Displays Area Page Maps or a specific Area Page Map
- **-v** – Displays map blocks.
- **-x** – Displays XPT fields.
- **-f** – Prompts for the *sid* or *idx* for which the XPT fields are to be displayed.
- *sid* – Specifies the segment ID. Symbols, hexadecimal values, or hexadecimal expressions may be used for this argument.
- *idx* – Specifies the index for the specified area. Symbols, hexadecimal values, or hexadecimal expressions can be used for this argument.

The optional arguments listed above determine the data that is displayed. Summary information is displayed when no parameter is provided.

## Aliases

No aliases.

## Example

The following is an example of how to use the **pta** subcommand:

```
KDB(0)> pta -?
VMM PTA segment (1) @ C0000000
Usage: pta
  pta -r[oot] [sid] [seg no.]  /to print XPT root
  pta -d[blk] [sid] [seg no.]  /to print XPT direct blocks
  pta -a[pm]  [idx] [seg no.]  /to print Area Page Maps
  pta -apmno apmidx  segno     /to print specific APM
  pta -v[map] [idx] [seg no.]  /to print map blocks
  pta -x[pt]   xpt  /to print XPT fields
  pta -f[ind] (prompt for sid/pno)  /to find or print XPT fields
KDB(0)> pta
VMM PTA segment (1) @ C0000000
VMM PTA segment @ C0000000
pta_root....... @ C0000000  pta_hiapm...... : 00000200
pta_vmapfree... : 0000CE46  pta_usecount... : 00040000
pta_anchor(0).. : 000000E5  pta_anchor(1).. : 00000000
pta_anchor(2).. : 00000000  pta_anchor(3).. : 00000000
pta_anchor(4).. : 00000000  pta_anchor(5).. : 00000000
pta_freecnt.... : 00000008  pta_freetail... : 000001FF
pta_apm(1rst).. @ C0000600  pta_xptdblk.... @ C0080000
KDB(0)>
```

## pte subcommand

### Purpose

The **pte** subcommand provides options for displaying information about the VMM page table entries.

### Syntax

**pte** [*menu options*]

### Parameters

- *menu options* – Use menu options and parameters with the subcommand to avoid the display of menus and prompts.

If the **pte** subcommand is invoked without parameters, menus and prompts are used to determine the data to be displayed. If the menu selections and required values are known, you can use them as subcommand parameters.

### Aliases

No aliases.

### Example

The following is an example of how to use the **pte** subcommand:

```
KDB(0)> pte
VMM PTE
Select the PTE to display by:
 1) index
 2) sid,pno
 3) page frame
 4) PTE group
Enter your choice: 2
Enter the sid (in hex): 400
Enter the pno (in hex): 0

 PTEX  v      SID      h  avpi      RPN      r c wimg pp L pin
002001 1 0000000000400 0   00  0000000021E36 1 0 0002 01 0  0
KDB(0)> pte 4
Enter the sid (in hex): 400
Enter the pno (in hex): 0

 PTEX  v      SID      h  avpi      RPN      r c wimg pp L pin
002000 1 0000000000000 0   00  0000000000400 1 0 0002 00 0  0
002001 1 0000000000400 0   00  0000000021E36 1 0 0002 01 0  0
002002 0 0000000000000 0   00  0000000000000 0 0 0000 00 0  0
002003 0 0000000000000 0   00  0000000000000 0 0 0000 00 0  0
002004 0 0000000000000 0   00  0000000000000 0 0 0000 00 0  0
002005 0 0000000000000 0   00  0000000000000 0 0 0000 00 0  0
002006 0 0000000000000 0   00  0000000000000 0 0 0000 00 0  0
002007 0 0000000000000 0   00  0000000000000 0 0 0000 00 0  0

 PTEX  v      SID      h  avpi      RPN      r c wimg pp L pin
1FDFF8 0 0000000000000 0   00  0000000000000 0 0 0000 00 0  0
1FDFF9 0 0000000000000 0   00  0000000000000 0 0 0000 00 0  0
1FDFFA 0 0000000000000 0   00  0000000000000 0 0 0000 00 0  0
1FDFFB 0 0000000000000 0   00  0000000000000 0 0 0000 00 0  0
1FDFFC 0 0000000000000 0   00  0000000000000 0 0 0000 00 0  0
1FDFFD 0 0000000000000 0   00  0000000000000 0 0 0000 00 0  0
1FDFFE 0 0000000000000 0   00  0000000000000 0 0 0000 00 0  0
```

```
1FDFFF 0 0000000000000 0   00  0000000000000 0 0 0000 00 0  0
KDB(0)>
```

# rmap subcommand

## Purpose

The **rmap** subcommand displays the real address range mapping table.

## Syntax

**rmap** [*] [*slot*]

## Parameters

- * – Displays all real address range mappings.
- *slot* – Displays the real address range mapping for the specified slot. This value must be a hexadecimal value.

If the asterisk ( * ) parameter is specified, a summary of all entries is displayed. If a slot number is specified, only that entry is displayed. If no parameter is specified, the user is prompted for a slot number, and data for that and all higher slots is displayed.

## Aliases

No aliases.

## Example

The following is an example of how to use the **rmap** subcommand:

```
KDB(0)> rmap *
            SLOT       RADDR       SIZE V ALIGN     <name>


vmrmap+000030 01 00000000000 000093534F 0 00000000 Kernel
vmrmap+000058 02 00007FAC000 0000008FEC 0 00000000 IPL control block
vmrmap+000080 03 00000936000 0000021000 0 00001000 MST
vmrmap+000120 07 00002000000 0001680000 0 00400000 s/w PFT
vmrmap+000148 08 00000C00000 0000180000 0 00400000 PVT
vmrmap+000170 09 00003680000 0001000000 0 00001000 PVLIST
vmrmap+000198 0A 00008000000 0002000000 0 02000000 PFT
vmrmap+0001C0 0B 00000957000 0000100000 0 00001000 s/w HAT
vmrmap+0001E8 0C 00000A57000 0000100000 0 00001000 APT
vmrmap+000210 0D 00000B57000 0000020000 0 00001000 AHAT
vmrmap+000238 0E 00000B77000 0000080000 0 00001000 RPT
vmrmap+000260 0F 00000D80000 0000020000 0 00001000 RPHAT
vmrmap+000288 10 00000DA0000 0000018000 0 00001000 PDT
vmrmap+0002B0 11 00000BF7000 0000001000 0 00001000 PTAR
vmrmap+0002D8 12 00000BF8000 0000002000 0 00001000 PTAD
vmrmap+000300 13 00000BFA000 0000003000 0 00001000 PTAI
vmrmap+000328 14 00000BFD000 0000001000 0 00001000 DMAP
vmrmap+0003A0 17 00000DB8000 0000020000 0 00001000 MEM_POOL & FRAME_SET
vmrmap+000468 1C 00000FE2000 000001E000 0 00000000 RMALLOC
vmrmap+000490 1D 00000BFE000 0000002000 0 00001000 VMINT
KDB(0)> rmap 11
RMAP entry 0011 of 004F: PTAR
> valid
> has mempool/frameset ids
Real address      : 0000000000BF7000
Effective address : 00000000C0000000
Size              : 0000000000001000
Alignment         : 0000000000001000
WIMG bits         : 2
vmpool requested  : 00
vmpool actual     : 00
KDB(0)> rmap
```

```
VMM usage: rmap [*][slot]
Enter the RMAP index (0-004F): 11
RMAP entry 0011 of 004F: PTAR
> valid
> has mempool/frameset ids
Real address      : 0000000000BF7000
Effective address : 00000000C0000000
Size              : 0000000000001000
Alignment         : 0000000000001000
WIMG bits         : 2
vmpool requested  : 00
vmpool actual     : 00
RMAP entry 0012 of 004F: PTAD
> valid
> has mempool/frameset ids
Real address      : 0000000000BF8000
Effective address : 00000000C0080000
Size              : 0000000000002000
Alignment         : 0000000000001000
WIMG bits         : 2
vmpool requested  : 00
vmpool actual     : 00
RMAP entry 0013 of 004F: PTAI
> valid
(0)> more (^C to quit) ?
> has mempool/frameset ids
Real address      : 0000000000BFA000
Effective address : 00000000C00C0000
Size              : 0000000000003000
Alignment         : 0000000000001000
WIMG bits         : 2
vmpool requested  : 00
vmpool actual     : 00
RMAP entry 0014 of 004F: DMAP
> valid
> has mempool/frameset ids
Real address      : 0000000000BFD000
Effective address : 00000000D0000000
Size              : 0000000000001000
Alignment         : 0000000000001000
WIMG bits         : 2
vmpool requested  : 00
vmpool actual     : 00
RMAP entry 0015 of 004F: unknown
RMAP entry 0016 of 004F: unknown
<snip>
```

# rvsid subcommand

## Purpose

The **rvsid** subcommand displays reserved vsid information (struct rvsid_data).

**Note:** The **rvsid** subcommand is only supported when you use the **kdb** command or the KDB kernel debugger on the 64-bit kernel.

## Syntax

**rvsid**

## Parameters

No parameters are supported.

## Aliases

No aliases.

## Example

The following is an example of how to use the **rvsid** subcommand:

```
(0)> rvsid

Reserved Vsid Control Variables: 000000000023D4E0rvsid_da+000000

num lgpg vsids per group  (lgpg_vsids_per_group) : 00000006
use spec. lgpg vsid alloc (lgpg_vsid_on)         : 00000000
rsvd vsid alloc interval  (sid_int)              : 00000200
number of reserved vsids  (num_vsids)            : 00000000
highest reserved vsid     (hi_vsid)              : 00000000
highest reserved sidx+1   (hi_sidx)              : 00000000
num reserved vsids in use (num_inuse)            : 00000000
reserved vsids high water (hi_inuse)             : 00000000
(0)>
```

## scb subcommand

### Purpose

The **scb** subcommand provides options for display of information about VMM segment control blocks.

### Syntax

**scb** [*menu options*]

### Parameters

- *menu options* – Use menu options and parameters with the subcommand to avoid display of menus and prompts.

If this subcommand is invoked without parameters, then menus and prompts are used to determine the data that is displayed. If the menu selections and required values are known, you can use them as subcommand parameters.

### Aliases

No aliases.

### Example

The following is an example of how to use the **scb** subcommand:

```
KDB(0)> scb
VMM SCBs
Select the scb to display by:
 1) index
 2) sid
 3) srval
 4) search on sibits
 5) search on npsblks
 6) search on nvpages
 7) search on npages
 8) search on npseablks
 9) search on lock
 a) search on segment type
 b) add total scb_vpages
 c) search on segment class
 d) search on segment pvproc
Enter your choice: 2
Enter the sid (in hex): 00024012

VMM SCB Addr B04775F4 Index 00000012 of 0000050B  Segment ID: 00024012

WORKING STORAGE SEGMENT
> (_segtype)..... working segment
> (_defd)........ deferred disk alloc
> (_privseg)..... process private segment
> (_compseg)..... computational segment
> (_privatt)..... process attachment
segment info bits        (_sibits)  : 88408800
default storage key      (_defkey)  : 2
extent of growing down   (minvpn)   : 0000FF3A 65338
last page user region    (sysbr)    : FFFFFFFF    -1
up limit                 (uplim)    : 00000000     0
down limit               (downlim)  : 0000EF23 61219
number of pgsp blocks    (npsblks)  : 00000000     0
number of virtual pages  (vpages)   : 00000004     4
freeze count             (frozen)   : 00000000     0
number of epsa blocks    (npseablks): 00000000     0
XPT root seg number      (xptrseg)  : 00000002     2
```

```
offset of XPT root       (xptroff)  : 00000302   770
XPT root address                    : C00C0800
(0)> more (^C to quit) ?
class ID                 (classid)  : 00000000     0
physical attachments     (_att)     : 00000000
mmap reference count     (refcnt)   : 00000000
pvproc ptr & pid                    : E2000400 00000204
mempools                            : 0000000000000000
non-fblu pageout count   (npopages) : 0000
xmem attach count        (xmemcnt)  : 0000
pages in real memory     (npages)   : 00000004
pinned pages in memory   (npinpages): 00000003
lru pageout count        (npopages) : 00000000
proc pointer             (proc)     : E2000400
page frame at head       (sidlist)  : 00000FCF
max assigned page number (maxvpn)   : FFFFFFFF
lock                     (lock)     :@B047764C 00000000
KDB(0)> scb
VMM SCBs
Select the scb to display by:
 1) index
 2) sid
 3) srval
 4) search on sibits
 5) search on npsblks
 6) search on nvpages
 7) search on npages
 8) search on npseablks
 9) search on lock
 a) search on segment type
 b) add total scb_vpages
 c) search on segment class
 d) search on segment pvproc
Enter your choice: 7

Find all scbs whose npages is greater than (in hex):2000

VMM SCB Addr B04774E0 Index 0000000F of 0000050B  Segment ID: 0001E00F

WORKING STORAGE SEGMENT
> (_segtype)..... working segment
> (_defd)........ deferred disk alloc
> (_system)...... system segment
> (_compseg)..... computational segment
segment info bits        (_sibits)  : 88088000
default storage key      (_defkey)  : 2
extent of growing down   (minvpn)   : 00010000 65536
up limit                 (uplim)    : 0000FFFF 65535
down limit               (downlim)  : 00010000 65536
number of pgsp blocks    (npsblks)  : 00000000     0
number of virtual pages  (vpages)   : 000030F8 12536
freeze count             (frozen)   : 00000000     0
number of epsa blocks    (npseablks): 00000000     0
XPT root seg number      (xptrseg)  : 00000001     1
offset of XPT root       (xptroff)  : 00000333   819
XPT root address                    : C00CCC00
class ID                 (classid)  : 00000000     0
physical attachments     (_att)     : 00000000
(0)> more (^C to quit) ?
mmap reference count     (refcnt)   : 00000000
non-fblu pageout count   (npopages) : 0000
xmem attach count        (xmemcnt)  : 0015
pages in real memory     (npages)   : 000030F8
pinned pages in memory   (npinpages): 00000CD4
lru pageout count        (npopages) : 00000000
proc pointer             (proc)     : 0028F908
page frame at head       (sidlist)  : 0005F2E0
```

```
max assigned page number (maxvpn)   : 000038A2
lock                     (lock)     :@B0477538 00000000

00000001 (hex) matches found with npages > 00002000.
KDB(0)> scb 1
Enter the index (in hex): 0000000F

VMM SCB Addr B04774E0 Index 0000000F of 0000050B  Segment ID: 0001E00F

WORKING STORAGE SEGMENT
> (_segtype)..... working segment
> (_defd)........ deferred disk alloc
> (_system)...... system segment
> (_compseg)..... computational segment
segment info bits        (_sibits)  : 88088000
default storage key      (_defkey)  : 2
extent of growing down   (minvpn)   : 00010000 65536
up limit                 (uplim)    : 0000FFFF 65535
down limit               (downlim)  : 00010000 65536
number of pgsp blocks    (npsblks)  : 00000000     0
number of virtual pages  (vpages)   : 000030F8 12536
freeze count             (frozen)   : 00000000     0
number of epsa blocks    (npseablks): 00000000     0
XPT root seg number      (xptrseg)  : 00000001     1
offset of XPT root       (xptroff)  : 00000333   819
XPT root address                    : C00CCC00
class ID                 (classid)  : 00000000     0
physical attachments     (_att)     : 00000000
(0)> more (^C to quit) ?
mmap reference count     (refcnt)   : 00000000
non-fblu pageout count   (npopages) : 0000
xmem attach count        (xmemcnt)  : 0015
pages in real memory     (npages)   : 000030F8
pinned pages in memory   (npinpages): 00000CD4
lru pageout count        (npopages) : 00000000
proc pointer             (proc)     : 0028F908
page frame at head       (sidlist)  : 0005F2E0
max assigned page number (maxvpn)   : 000038A2
lock                     (lock)     :@B0477538 00000000
KDB(0)>
```

## segst64 subcommand

### Purpose

The **segst64** subcommand displays the segment state information for a 64-bit process.

### Syntax

**segst64** [**-p** *pid* | **-e** *esid* | **-s** *seg* | *value*]

### Parameters

- **-p** *pid* – Specifies the process ID of a 64-bit process. This must be a decimal or hexadecimal value depending on the setting of the **hexadecimal_wanted** switch.
- **-e** *esid* – Specifies the first segment register to display. The lower register numbers 0, 1, and 2 are ignored. This parameter must be a hexadecimal value.
- **-s** *seg* – Limits the display to only segment register with a segment state that matches *seg*. Possible values for *seg* are: SEG_AVAIL, SEG_SHARED, SEG_MAPPED, SEG_MRDWR, SEG_DEFER, SEG_MMAP, SEG_WORKING, SEG_RMMAP, SEG_OTHER, SEG_EXTSHM, and SEG_TEXT.
- *value* – Sets the limit to display only segments with the specified value for the **segfileno** field. This value must be hexadecimal.

### Aliases

No aliases.

### Example

The following is an example of how to use the **segst64** subcommand:

```
KDB(0)> segst64  //display
snode    base     last     nvalid   sfwd     sbwd
00000000 00000003 FFFFFFFE 00000010 00000001 FFFFFFFF
ESID           segstate segflag num_segs fno/shmp/srval/nsegs
SR00000003>[ 0]       SEG_AVAIL 00000000 0000000A
SR0000000D>[ 1]       SEG_OTHER 00000001 00000001
SR0000000E>[ 2]       SEG_AVAIL 00000000 00000001
SR0000000F>[ 3]       SEG_OTHER 00000001 00000001
SR00000010>[ 4]        SEG_TEXT 00000001 00000001
SR00000011>[ 5]     SEG_WORKING 00000001 00000000
SR00000012>[ 6]       SEG_AVAIL 00000000 8000FFF8
SR8001000A>[ 7]     SEG_WORKING 00000001 00000000
SR8001000B>[ 8]       SEG_AVAIL 00000000 00010009
SR80020014>[ 9]     SEG_WORKING 00000001 00000000
SR80020015>[10]       SEG_AVAIL 00000000 0FFDFFEA
SR8FFFFFFF>[11]     SEG_WORKING 00000001 00000000
SR90000000>[12]        SEG_TEXT 00000001 00000001
SR90000001>[13]       SEG_AVAIL 00000000 0FFFFFFE
SR9FFFFFFF>[14]        SEG_TEXT 00000001 00000001
SRA0000000>[15]       SEG_AVAIL 00000000 5FFFFFFF
snode    base     last     nvalid   sfwd     sbwd
00000001 FFFFFFFF FFFFFFFF 00000001 FFFFFFFF 00000000
ESID           segstate segflag num_segs fno/shmp/srval/nsegs
SRFFFFFFFF>[ 0]     SEG_WORKING 00000001 00000000
```

## sr64 subcommand

### Purpose

The **sr64** subcommand displays segment registers for a 64-bit process for 32-bit and 64-bit kernels.

### Syntax

**sr64** [**-p** *pid*] [*esid*] [*size*] // for 32 bit kernel

**sr64** [**-g** *range size*] [**-p** *pid*] [*esid*] [*size*] // for 64 bit kernel

### Parameters

- **-p** *pid* – Specifies the process ID of a 64-bit process. This must be a decimal or hexadecimal value, depending on the setting of the *hexadecimal_wanted* switch. The *hexadecimal_wanted* switch is changed using the **set** subcommand.
- **-g** – Displays esids from the global system address space. The minimum range size and the default is `3`, but a larger range can optionally be provided.

  **Note:** The **-g** flag is only supported for the 64-bit kernel.
- *esid* – Specifies the first segment register to display. Register numbers lower than the specified register are ignored. This parameter must be a hexadecimal value.
- *size* – Specifies the value to be added to the first segment register to determine the last segment register to display. This parameter must be a hexadecimal value.

If no parameters are specified, the current process is used. Another process can be specified by using the **-p** *pid* flag. Additionally, the *esid* and *size* parameters can be used to limit the segment registers displayed. The *esid* value determines the first segment register to display. The value of *esid* + *size* determines the last segment register to display.

The registers are displayed in groups of 16. If necessary, the value of the *esid* parameter is rounded down to a multiple of 16, and the *size* is rounded up to a multiple of 16. For example: `sr64 11 11` displays the segment registers `10` through `2f`.

### Aliases

No aliases.

### Example

The following is an example of how to use the **sr64** subcommand for a 32-bit kernel:

```
KDB(0)> sr64 ?  //display help
Usage: sr64 [-p pid] [esid] [size]
KDB(0)> sr64  //display all segment registers
SR00000000: 60000000  SR00000002: 60002B45  SR0000000D: 6000614C
SR00000010: 6000520A  SR00000011: 6000636C
SR8001000A: 60003B47
SR80020014: 6000B356
SR8FFFFFFF: 60000340
SR90000000: 60001142
SR9FFFFFFF: 60004148
SRFFFFFFFF: 6000B336
KDB(0)> sr64 11  //display up to 16 SRs from 10
Segment registers for address space of Pid: 000048CA
SR00000010: 6000E339  SR00000011: 6000B855
KDB(0)> sr64 0 100  //display up to 256 SRs from 0
Segment registers for address space of Pid: 000048CA
SR00000000: 60000000  SR00000002: 60002B45  SR0000000D: 6000614C
SR00000010: 6000520A  SR00000011: 6000636C
```

The following is an example of how to use the **sr64** subcommand for a 64-bit kernel:

```
KDB(0)> sr64 -g
Segment registers for global address space
kernel..... sr000000000: 00000400
vmm data... srF10000004: 00801400
vmm pta.... srF10000005: 01002400
vmm diskmap srF10000006: 01803400 srF10000007: 02004400 srF10000008: 02805400 ..

vmm ame.... srF1000000A: 03807400 srF1000000B: 04008400 srF1000000C: 04809400 ..

vmm scb.... srF1000000E: 0580B400 srF1000000F: 0600C400 srF10000010: 0680D400 ..

vmm swhat.. srF100000BE: 0D8BB400 srF100000BF: 0E0BC400 srF100000C0: 0E8BD400 ..

real heap.. srF1000013E: 0D93B400 srF1000013F: 0E13C400 srF10000140: 0E93D400 ..

proc-thread srF10000878: 0A075400 srF10000879: 0B876400 srF1000087A: 0B077400 ..

mbuf....... srF1000089C: 0C099400 srF1000089D: 0D89A400 srF1000089E: 0D09B400 ..

ldr........ srF100009A0: 0E19D400 srF100009A1: 0F99E400 srF100009A2: 0F19F400 ..

jfs lkword. srF100009C0: 0E1BD400
kernel heap srF10000F00: 0E6FD400 srF10000F01: 0FEFE400 srF10000F02: 0F6FF400 ..

global ext. srF100009F2: 071EF400 srF100009F3: 089F0400 srF100009F4: 081F1400 ..

global lgpg srF10000EE0: 0E6DD400 srF10000EE1: 0FEDE400 srF10000EE2: 0F6DF400 ..

vmm ksp.... srF20001001: 10010001400 srF20001002: 10020002400 srF20001003: 10030
003400 ..
KDB(0)> sr64 -g 6
Segment registers for global address space
kernel..... sr000000000: 00000400
vmm data... srF10000004: 00801400
vmm pta.... srF10000005: 01002400
vmm diskmap srF10000006: 01803400 srF10000007: 02004400 srF10000008: 02805400
vmm diskmap srF10000009: 03006400
vmm ame.... srF1000000A: 03807400 srF1000000B: 04008400 srF1000000C: 04809400
vmm ame.... srF1000000D: 0500A400
vmm scb.... srF1000000E: 0580B400 srF1000000F: 0600C400 srF10000010: 0680D400
vmm scb.... srF10000011: 0700E400 srF10000012: 0780F400 srF10000013: 08010400 ..

vmm swhat.. srF100000BE: 0D8BB400 srF100000BF: 0E0BC400 srF100000C0: 0E8BD400
vmm swhat.. srF100000C1: 0F0BE400 srF100000C2: 0F8BF400 srF100000C3: 000C0400 ..

real heap.. srF1000013E: 0D93B400 srF1000013F: 0E13C400 srF10000140: 0E93D400
real heap.. srF10000141: 0F13E400
proc-thread srF10000878: 0A075400 srF10000879: 0B876400 srF1000087A: 0B077400
proc-thread srF1000087B: 0C878400 srF1000087C: 0C079400 srF1000087D: 0D87A400 ..

mbuf....... srF1000089C: 0C099400 srF1000089D: 0D89A400 srF1000089E: 0D09B400
mbuf....... srF1000089F: 0E89C400 srF100008A0: 0E09D400 srF100008A1: 0F89E400 ..

ldr........ srF100009A0: 0E19D400 srF100009A1: 0F99E400 srF100009A2: 0F19F400
ldr........ srF100009A3: 009A0400 srF100009A4: 001A1400 srF100009A5: 019A2400 ..

jfs lkword. srF100009C0: 0E1BD400
kernel heap srF10000F00: 0E6FD400 srF10000F01: 0FEFE400 srF10000F02: 0F6FF400
kernel heap srF10000F03: 00F00400 srF10000F04: 00701400 srF10000F05: 01F02400 ..

global ext. srF100009F2: 071EF400 srF100009F3: 089F0400 srF100009F4: 081F1400
global ext. srF100009F5: 099F2400 srF100009F6: 091F3400 srF100009F7: 0A9F4400 ..

global lgpg srF10000EE0: 0E6DD400 srF10000EE1: 0FEDE400 srF10000EE2: 0F6DF400
global lgpg srF10000EE3: 00EE0400 srF10000EE4: 006E1400 srF10000EE5: 01EE2400 ..
```

```
vmm ksp.... srF20001001: 10010001400 srF20001002: 10020002400 srF20001003: 10030
003400
vmm ksp.... srF20001004: 10040004400 srF20001005: 10050005400 srF20001006: 10060
006400 ..
KDB(0)>
```

## ksp subcommand

### Purpose

The **ksp** subcommand displays information about the Kernel Special Purpose (KSP) region.

**Note:** Because some of the contents of the KSP region depend upon whether you are using the 32-bit or 64-bit kernel, the output of the **ksp** subcommand varies slightly depending upon which kernel you use.

### Syntax

**ksp**

### Parameters

No parameters.

### Aliases

No aliases.

### Example

The following is an example of how to use the **ksp** subcommand:

```
KDB(0)> ksp

Kernel Special Purpose (KSP) Region Info

KSP_FIRST_SID........010000000
KSP_SID_BASE.........010010001
KSP_SIDX_BASE........010010001
KSP_SIDHASH_INC.......00010001
KSP_REGION_INC.......010000000
KSP_SID_END..........02D830D83
KSP_ESID_BASE........F20001001
KSP_ESID_END.........F20003000
KSP_TOTAL_SIDS........00000D82
KSP_ARCH_NUMSIDS......00000D82

Data Structures in the KSP Region:

VMM SWPFT Address............F200010010000000 vmmswpft+000000
VMM SWPFT Esid Range.........F20001001, F20001001
VMM SWPFT Start (sidx,sid)...010010001, 010010001
VMM SWPFT End (sidx,sid).....010010001, 010010001
VMM SWPFT Size in #Segments..(partial segment)

VMM HWPFT Address............0000000000000000
VMM HWPFT Esid Range.........000000000, 000000000
VMM HWPFT Start (sidx,sid)...000000000, 000000000
VMM HWPFT End (sidx,sid).....000000000, 000000000
VMM HWPFT Size in #Segments..00000001

VMM PVT Address............F200010020000000
VMM PVT Esid Range.........F20001002, F20001002
VMM PVT Start (sidx,sid)...010020002, 010020002
VMM PVT End (sidx,sid).....010020002, 010020002
VMM PVT Size in #Segments..(partial segment)

VMM PVLIST Address............F200020030000000
VMM PVLIST Esid Range.........F20002003, F20002003
VMM PVLIST Start (sidx,sid)...020030003, 020030003
VMM PVLIST End (sidx,sid).....020030003, 020030003
```

```
VMM PVLIST Size in #Segments..(partial segment)

Segment ID and related definitions for reference

NUMSIDS...............10000000
VM_L2_MAXARCH_VSID....00000025
VM_MAXARCH_VSID.......1FFFFFFFFF
VM_L2_IOSID_BIT.......00000024
IOSIDBIT..............1000000000
IOSIDMASK.............FFFFFFFFF
GLOB_ESID_LAST........F10000FFF

(0)>
```

## ste subcommand

### Purpose

The **ste** subcommand provides options for displaying information about segment table entries for 64-bit processes.

### Syntax

**ste** [**-p** *pid*] [*menu options*]

### Parameters

- **-p** *pid* – Specifies the process identifier to switch to before the menu is invoked. If this optional flag is omitted, the current process is assumed.
- *menu options* – Enter menu options and parameters along with the subcommand to avoid displaying menus and prompts. If you do not enter menu options, the menu is invoked.

If this subcommand is invoked without parameters, then menus and prompts are used to determine the data to display.

### Aliases

No aliases.

### Example

The following is an example of how to use the **ste** subcommand:

```
KDB(0)> ste -p 042B8
Switch to proc: E2008400
Segment Table (STAB)
Select the STAB entry to display by:
 1) esid
 2) sid
 3) dump hash class (input=esid)
 4) dump entire stab
Enter your choice: 4
0000000022821000: ESID 0000000090000000 VSID 00000000000041A2 V Ks Kp
0000000022821010: ESID 0000000000000000 VSID 0000000000000000 V Ks Kp
0000000022821020: ESID 0000000000000000 VSID 0000000000000000
0000000022821030: ESID 0000000000000000 VSID 0000000000000000
0000000022821040: ESID 0000000000000000 VSID 0000000000000000
0000000022821050: ESID 0000000000000000 VSID 0000000000000000
0000000022821060: ESID 0000000000000000 VSID 0000000000000000
0000000022821070: ESID 0000000000000000 VSID 0000000000000000
0000000022821080: ESID 0000000000000000 VSID 0000000000000000
0000000022821090: ESID 0000000000000000 VSID 0000000000000000
00000000228210A0: ESID 0000000000000000 VSID 0000000000000000
00000000228210B0: ESID 0000000000000000 VSID 0000000000000000
00000000228210C0: ESID 0000000000000000 VSID 0000000000000000
00000000228210D0: ESID 0000000000000000 VSID 0000000000000000
00000000228210E0: ESID 0000000000000000 VSID 0000000000000000
00000000228210F0: ESID 0000000000000000 VSID 0000000000000000
0000000022821100: ESID 0000000000000002 VSID 0000000000010488 V Ks Kp
0000000022821110: ESID 0000000000000000 VSID 0000000000000000
0000000022821120: ESID 0000000000000000 VSID 0000000000000000
0000000022821130: ESID 0000000000000000 VSID 0000000000000000
0000000022821140: ESID 0000000000000000 VSID 0000000000000000
0000000022821150: ESID 0000000000000000 VSID 0000000000000000
(0)> more (^C to quit) ?
<snip>
KDB(0)> ste
Segment Table (STAB)
```

```
Select the STAB entry to display by:
 1) esid
 2) sid
 3) dump hash class (input=esid)
 4) dump entire stab
Enter your choice: 3
Hash Class to dump (in hex) [esid ok here]: 10
          PRIMARY HASH GROUP
0000000022821800: ESID 0000000000000010 VSID 0000000000000400 V Ks Kp
0000000022821810: ESID 0000000000000000 VSID 0000000000000000
0000000022821820: ESID 0000000000000000 VSID 0000000000000000
0000000022821830: ESID 0000000000000000 VSID 0000000000000000
0000000022821840: ESID 0000000000000000 VSID 0000000000000000
0000000022821850: ESID 0000000000000000 VSID 0000000000000000
0000000022821860: ESID 0000000000000000 VSID 0000000000000000
0000000022821870: ESID 0000000000000000 VSID 0000000000000000
          SECONDARY HASH GROUP
0000000022821780: ESID 0000000000000000 VSID 0000000000000000
0000000022821790: ESID 0000000000000000 VSID 0000000000000000
00000000228217A0: ESID 0000000000000000 VSID 0000000000000000
00000000228217B0: ESID 0000000000000000 VSID 0000000000000000
00000000228217C0: ESID 0000000000000000 VSID 0000000000000000
00000000228217D0: ESID 0000000000000000 VSID 0000000000000000
00000000228217E0: ESID 0000000000000000 VSID 0000000000000000
00000000228217F0: ESID 0000000000000000 VSID 0000000000000000
KDB(0)> ste 1
Enter the esid (in hex): 0FFFFFFFF
0000000022821FA0: ESID 00000000FFFFFFFF VSID 00000000000263F3 V Ks Kp
KDB(0)>
```

## vmbufst subcommand

### Purpose

The **vmbufst** subcommand displays VMM **buf** structures.

### Syntax

**vmbufst** [*bufaddr*]

### Parameters

- *bufaddr* – Specifies the address of the **buf** structure to display. If the parameter is omitted, you are prompted to enter it.

The **vmbufst** subcommand is similar to the general filesystem **buf** subcommand. It displays a subset of the fields and automatically traverses any `buf.av_forw` chain.

### Aliases

No aliases.

### Example

The following is an example of how to use the **vmbufst** subcommand:

```
KDB(7)> vmbufst
Enter address of the bufst:34DD79F0   //entered 34DD79F0> vmbufst 34DD79F0
flags.......: 000C8001
b_forw......: 00000000    b_back..... : 00000000
av_forw.....: 00000000    av_back.....: 00000000
iodone......: 020B0A0C    b_vp........: 00000000
b_dev.......: 000E0003    b_blkno.....: 01B82700
b_addr......: 00000000    b_bcount....: 00001000
b_error.....: 00          xmem is at  : 00504C78

KDB(7)> buf 34DD79F0               // contrast with the buf cmd
                  DEV    VNODE    BLKNO FLAGS

  0 34DD79F0 000E0003 00000000 01B82700 READ SPLIT MPSAFE INITIAL
forw     00000000 back     00000000 av_forw  00000000 av_back  00000000
addr     00000000 blkno    01B82700
vp       00000000 flags    000C8001 bcount   00001000 resid    00000000
work     34E4B000 error    00000000 options  00000000 event    FFFFFFFF
iodone:  020B0A0C
start.tv_sec       00015947 start.tv_nsec      00000000
xmemd.aspace_id    FFFFFFFC xmemd.prexflags    00000011
xmemd.orig_xmem    34DF0030 xmemd.rlist        34DF1030
orig.aspace_id     00000000 orig.subspace_id   008384CE
orig.subspace_id2  00000000 orig.uaddr         00000000

KDB(7)>

Another difference between the two commands is that the vmbufst command
automatically traverses any av_forw list:
KDB(0)> buf @r5
                                  DEV    BLKNO FLAGS

  0 F10000AFD0024F00 8000000D00000001 00DE27F0 MPSAFE INITIAL
forw     0000000000000000 back     0000000000000000
av_forw  F10000AFD002A780 av_back  0000000000000000
addr     0000000000008000 blkno    0000000000DE27F0
vp       0000000000000000 flags    00000000000C0000
bcount   0000000000002000 resid    0000000000000000
work     0000000000000001 error    00000000
```

```
options  00000000 event    FFFFFFFFFFFFFFFF
iodone:  034CD180
start.tv_sec       00000000401F4D2B start.tv_nsec      00000000
xmemd.aspace_id    00000000         xmemd.num_sids     00000001
xmemd.subspace_id  00010001914D9000 xmemd.vaddr        0000000000000000
xmemd.prexflags    00000013         xmemd.xp@          F10000AFD0024FB0
xmemd.xp.total     0000000000000020 xmemd.xp.used      0000000000000002
xmemd.xp.s_vpn     0000000000000008 xmemd.xp.rpn       F100009E25733000
KDB(0)> vmbufst @r5                       <also displays the buf at F10000AFD002A780>
flags.......: 00000000000C0000
b_forw......: 0000000000000000    b_back..... : 0000000000000000
av_forw.....: F10000AFD002A780    av_back.....: 0000000000000000
iodone......: 00000000034CD180    b_vp........: 0000000000000000
b_dev.......: 8000000D00000001    b_blkno.....: 0000000000DE27F0
b_addr......: 0000000000008000    b_bcount....: 0000000000002000
b_error.....: 00         xmem is at  : 0000000003016BB0

flags.......: 00000000000C0000
b_forw......: 0000000000000000    b_back..... : 0000000000000000
av_forw.....: 0000000000000000    av_back.....: 0000000000000000
iodone......: 00000000034CD180    b_vp........: 0000000000000000
b_dev.......: 8000000D00000001    b_blkno.....: 0000000000DE2800
b_addr......: 000000000000A000    b_bcount....: 0000000000002000
b_error.....: 00         xmem is at  : 0000000003016BB0

KDB(0)>
```

## vmaddr subcommand

### Purpose

The **vmaddr** subcommand displays addresses of **VMM** structures.

### Syntax

**vmaddr**

### Parameters

No parameters.

### Aliases

No aliases.

### Example

The following is an example of how to use the **vmaddr** subcommand:

```
KDB(0)> vmaddr

VMM Addresses

H/W PTE    : 0000000008000000 [real address]
H/W PVT    : 0000000000C00000 [real address]
H/W PVLIST : 0000000003680000 [real address]
S/W HAT    : A0000000vmmswhat+000000
S/W PFT    : 40000000vmmswpft+000000
AHAT       : B02A0000vmmdseg +2A0000
APT        : B02C0000vmmdseg +2C0000
RPHAT      : B03C0000vmmdseg +3C0000
RPT        : B03E0000vmmdseg +3E0000
PDT        : B0460000vmmdseg +460000
PFHDATA    : B0476000vmmdseg +476000
LOCKANCH   : D0000000lkwseg  +000000
SCBs       : B0476F7Cvmmdseg +476F7C
ESCBs      : BBC76F7Cvmmdseg+BC76F7C
LOCKWORDS  : D000B000lkwseg  +00B000
AMEs       : D0000000ameseg  +000000
LOCK:
 PMAP      : 00000000 00000000
KDB(0)>
```

# vmdmap subcommand

## Purpose

The **vmdmap** subcommand displays VMM disk maps.

## Syntax

**vmdmap** [*slot* | *Address*]

## Parameters

- *slot* – Specifies the Page Device Table (pdt) slot number. This parameter must be a decimal value.
- *address* – Specifies the address of a specific **vmdmap** structure to display.

If no parameters are entered, all paging and file system disk maps are displayed. To view a single disk map, enter a slot number.

## Aliases

No aliases.

## Example

The following is an example of how to use the **vmdmap** subcommand:

```
KDB(0)> vmdmap
PDT slot [0000] Vmdmap [D0000000] dmsrval [00006003]
mapsize................00020000 freecnt................0001FF55
agsize.................00000800 agcnt..................00000007
totalags...............00000040 lastalloc..............000000AA
maptype................00000003 clsize.................00000001
clmask.................00000080 version................00000000
btree..................00000000
btree_nxt..............00000000
agfree@................D0000030 tree@..................D00000A0
spare1@................D000002C mapsorsummary@.........D0000200
PDT slot [0011] Vmdmap [D0000000] dmsrval [00002081]
mapsize................00002000 freecnt................0000199B
agsize.................00000800 agcnt..................00000004
totalags...............00000004 lastalloc..............00000430
maptype................00000001 clsize.................00000008
clmask.................000000FF version................00000000
btree..................00000000
btree_nxt..............00000000
agfree@................D0000030 tree@..................D00000A0
spare1@................D000002C mapsorsummary@.........D0000200
PDT slot [0013] Vmdmap [D0000000] dmsrval [0003609B]
mapsize................00006000 freecnt................00004E2B
agsize.................00000800 agcnt..................00000008
totalags...............0000000C lastalloc..............000000DC
maptype................00000001 clsize.................00000020
clmask.................00000000 version................00000001
btree..................00000000
btree_nxt..............00000000
agfree@................D0000030 tree@..................D00000A0
spare1@................D000002C mapsorsummary@.........D0000200
<snip>
KDB(0)> vmdmap 11
PDT slot [0011] Vmdmap [D0000000] dmsrval [00002081]
mapsize................00002000 freecnt................0000199B
agsize.................00000800 agcnt..................00000004
totalags...............00000004 lastalloc..............00000430
maptype................00000001 clsize.................00000008
clmask.................000000FF version................00000000
```

```
btree..................00000000
btree_nxt..............00000000
agfree@................D0000030 tree@..................D00000A0
spare1@................D000002C mapsorsummary@.........D0000200
KDB(0)>
```

# vmint subcommand

## Purpose

The **vmint** subcommand displays VMM data for intervals.

## Syntax

**vmint** [ *base* | *list* | *range* ]

## Parameters

- *base* | *list* | *range* – Use one of these optional address input parameters. Identify a base of an interval array, the head of an interval, or the address of a range in an interval to be displayed.

  **Note:** The *base* and *range* parameters are typically only used for debugging problems in the vminterval code.

The **vmint** subcommand displays **VMM** structure vmintervals information. If no parameter is provided, information on system-wide intervals is displayed.

The **vmint** subcommand displays one of three types of information when an address input parameter is provided:

- If the address parameter is a base of an interval array, the entire array of vmintervals is displayed.
- If the address parameter is the head of an interval, the vminterval is displayed.
- If the address parameter is the address of one range in an interval, the specific range is displayed.

## Aliases

No aliases.

## Example

The following is an example of how to use the **vmint** subcommand:

```
KDB(0)> vmint

VMM vmint DATA:

VMINT_BADMEM: Memory holes            FFD90000 pages  lock @ 010B1420 00000000
        [270000,100000000)
VMINT_FIXCOM: Fixed common(BSS) memory        0000032F pages  lock @ 010B13E0 00000000
        [002937,002C65)
        [003A94,003A95)
VMINT_PINOBJ: PINNED object module      00001CF5 pages  lock @ 010B12E0 00000000
        [000000,000216)
        [000423,000427)
        [001000,001333)
        [00149C,002C44)
VMINT_PAGEDOBJ: PAGED object module     00000FA2 pages  lock @ 010B1320 00000000
        [0002BB,000410)
        [000428,00042B)
        [001463,00147E)
        [002C65,003A94)
VMINT_DBGOBJ:  DBG object module        00000326 pages  lock @ 010B1360 00000000
        [000216,0002BB)
        [000427,000428)
        [000485,0005B4)
        [001333,001463)
        [002C44,002C65)
VMINT_INITOBJ: INIT object module       00000023 pages  lock @ 010B13A0 00000000
        [000410,000423)
```

```
        [00042B,00042D)
        [00147E,00148B)
        [003A94,003A95)
VMINT_LGPG: Large page memory           00000000 pages  lock @ 012EC3C8 00000000
VMINT_FIXLMB: DR non-removeable memory  00019D8C pages  lock @ 010B16E0 00000000
        [000000,000A14)
        [000C14,000C18)
        [000C48,000C58)
        [001000,0015A7)
        [001800,002B80)
        [002C00,003C00)
        [00698B,0069AB)
        [007D2C,007D49)
        [008000,016A00)
        [017000,01F000)
```

```
KDB(0)> vmint 010B1418
       FFD90000 pages  lock @ 010B1420 00000000
        [270000,100000000)
```

```
KDB(0)> vmint 010B16B8
    [270000,100000000)              Prev: 010B1418  Next: 010B1438
KDB(0)> vmint 010B1438
        [FFFFFFFFFFFFFFFF,FFFFFFFFFFFFFFFF)             Prev: 010B16B8  Next: 010B1258
```

```
KDB(0)> vmint 010B1258
vminterval array based at 010B1258
        srad:   0000            freebase: 0
        freelist has 80 items starting with 010B1858
        freelist lock @ 010B12A0 00000000
        00001CF5 pages  lock @ 010B12E0 00000000
        [000000,000216)
        [000423,000427)
        [001000,001333)
        [00149C,002C44)
        00000FA2 pages  lock @ 010B1320 00000000
        [0002BB,000410)
        [000428,00042B)
        [001463,00147E)
        [002C65,003A94)
        00000326 pages  lock @ 010B1360 00000000
        [000216,0002BB)
        [000427,000428)
        [000485,0005B4)
        [001333,001463)
        [002C44,002C65)
        00000023 pages  lock @ 010B13A0 00000000
        [000410,000423)
        [00042B,00042D)
        [00147E,00148B)
        [003A94,003A95)
        0000032F pages  lock @ 010B13E0 00000000
        [002937,002C65)
        [003A94,003A95)
        FFD90000 pages  lock @ 010B1420 00000000
        [270000,100000000)
        00019D8C pages  lock @ 010B16E0 00000000
        [000000,000A14)
        [000C14,000C18)
        [000C48,000C58)
```

```
[001000,0015A7)
[001800,002B80)
[002C00,003C00)
[00698B,0069AB)
[007D2C,007D49)
[008000,016A00)
[017000,01F000)
```

## vmker subcommand

### Purpose

The **vmker** subcommand displays virtual memory kernel data.

### Syntax

**vmker** [**-pta**] [**-dr**] [**-seg**]

### Parameters

- **-pta** – Displays the Page Table Area (PTA) data.
- **-dr** – Displays dynamic memory reconfiguration related data.
- **-seg** – Displays VMM segment data.

General VMM kernel data is displayed when no parameter is supplied. All three flags are optional.

### Aliases

No aliases.

### Example

The following is an example of how to use the **vmker** subcommand:

```
KDB(1)> vmker

VMM Kernel Data:
        (use [-pta | -dr | -seg] for specific info)

rsvd pgsp blks    (psrsvdblks) : 00000200
total page frames (nrpages)    : 00280000
bad page frames   (badpages)   : 00000009
good page frames  (goodpages)  : 00280000
ipl page frames   (iplpages)   : 00280000
total pgsp blks   (numpsblks)  : 00020000
free pgsp blks    (psfreeblks) : 0001FE08
rsvd page frames  (pfrsvdblks) : 00080000
fetch protect     (nofetchprot): 00000000
max file pageout  (maxpout)    : 00000000
min file pageout  (minpout)    : 00000000
repage table size (rptsize)    : 00010000
next free in rpt  (rptfree)    : 00000000
repage decay rate (rpdecay)    : 0000005A
global repage cnt (sysrepage)  : 00000000
swhashmask        (swhashmask) : 001FFFFF
cachealign        (cachealign) : 00001000
overflows         (overflows)  : 00000000
reloads           (reloads)    : 00000247
compressed files  (noflush)    : 00000000
extended iplcb    (iplcbxptr)  : 0000000000000000
alias hash mask   (ahashmask)  : 0000FFFF
max pgs to delete (pd_npages)  : 00010000
vrld xlate hits   (vrldhits)   : 00000000
vrld xlate misses (vrldmisses) : 00000010
pgsp bufst waits (psbufwaitcnt): 00000000
fsys bufst waits (fsbufwaitcnt): 00000BB4
rsys bufst waits(rfsbufwaitcnt): 00000000
xpager bufst waits(xpagerbufwaitcnt): 00000000
phys_mem(s)      (phys_mem[0]) : 00280000
phys_mem(s)      (phys_mem[1]) : FFFFFFFF
phys_mem(s)      (phys_mem[2]) : 00000000
THRPGIO buf wait     (_waitcnt)  : 00000000
```

```
THRPGIO partial cnt (_partialcnt): 00000000
THRPGIO full cnt    (_fullcnt)   : 00000000
num lgpg's free'd   (nlgpgfreed) : 00000000
KDB(1)> vmker -pta

VMM PTA Related Data:

total pgsp blks   (numpsblks)  : 00020000
free pgsp blks    (psfreeblks) : 0001FE08
pta kproc tid   (ptakproc_tid) : 0002504B
# of ptasegments  (numptasegs) : 00000001
ptaseg(s)         (ptasegs[1]) : F100000050000000  sid:00020002  sidx:00000002
KDB(1)> vmker -seg

VMM Segment Related Data:

vmm srval         (vmmsrval)  : 10001400
ram disk srval    (ramdsrval) : 00000000
kernel ext srval  (kexsrval)  : 00000000
iplcb vsid        (iplcbvmh)  : 1F0FFF000
offset of iplcb   (iplcboff)  : 00000000
hashbits          (hashbits)  : 00000015
hashmask          (hashmask)  : 001FFFFF
hash shift amount (stoibits)  : 00000010
base config seg   (bconfsrval): 1E0FFE400
shadow srval      (ukernsrval): 00000000
kernel srval      (kernsrval) : 00000400
STOI/ITOS mask    (stoimask)  : 0000001F
STOI/ITOS sid mask(stoinio)   : 00000000
rmallocvmh        (rmallocvmh): 1B013B400
# of ptasegments  (numptasegs): 00000001
ptaseg(s)         (ptasegs[1]): F100000050000000
KDB(1)> vmker -dr

VMM DR Related Data:

total page frames    (nrpages) : 00280000
bad page frames     (badpages) : 00000009
good page frames   (goodpages) : 00280000
ipl page frames     (iplpages) : 00280000
rsvd page frames  (pfrsvdblks) : 00080000
DR mem adds          (addlmbs) : 00000000
DR mem removes        (rmlmbs) : 00000000
DR fixlmb migrates   (fixlmbs) : 00000000
DR reloads ena (ena_rldmigmiss): 00000000
DR reloads dis (dis_rldmigmiss): 00000000
DR refcntmiss  (migrefcntmiss) : 00000000
DR migr trans  (migtransients) : 00000000
DR mark trans (marktransients) : 00000000
DR migr misses  (vlookmigmiss) : 00000000
DR vmm migrates (vmm_migrates) : 00000000
DR serv migrates(serv_migrates): 00000000
DR vmpool adds      (add_vmps) : 00000000
DR vmpool removes   (rem_vmps) : 00000000
DR vmpool dormants  (dor_vmps) : 00000000
(1)> more (^C to quit) ?
DR mempool adds    (add_memps) : 00000000
DR mempool removes (rem_memps) : 00000000
DR mempool offline (off_memps) : 00000000
DR frameset adds    (add_frss) : 00000000
DR frameset removes (rem_frss) : 00000000
DR memory moves    (mem_moves) : 00000000
DR mempool       (rebal_calls) : 00000000
DR memp trans (memptransients) : 00000000
DR frs trans   (frstransients) : 00000000
KDB(1)>
```

## vmlocks subcommand

### Purpose

The **vmlocks** subcommand displays VMM spin lock data.

### Syntax

**vmlocks**

### Parameters

No parameters.

### Aliases

**vmlock**, **vl**

### Example

The following is an example of how to use the **vl** alias for the **vmlocks** subcommand:

```
KDB(0)> vl

GLOBAL LOCKS

pmap       lock at @ 00000000 FREE
vmap       lock at @ B0476100 FREE
ame        lock at @ B0476180 FREE
rpt global lock at @ B0476200 FREE
rpt pool lock [0] @ B0476280 FREE
rpt pool lock [1] @ B0476284 FREE
rpt pool lock [2] @ B0476288 FREE
rpt pool lock [3] @ B047628C FREE
rpt pool lock [4] @ B0476290 FREE
rpt pool lock [5] @ B0476294 FREE
rpt pool lock [6] @ B0476298 FREE
rpt pool lock [7] @ B047629C FREE
rpt pool lock [8] @ B04762A0 FREE
rpt pool lock [9] @ B04762A4 FREE
rpt pool lock [10] @ B04762A8 FREE
rpt pool lock [11] @ B04762AC FREE
rpt pool lock [12] @ B04762B0 FREE
rpt pool lock [13] @ B04762B4 FREE
rpt pool lock [14] @ B04762B8 FREE
(0)> more (^C to quit) ?
rpt pool lock [15] @ B04762BC FREE
alloc      lock at @ B0476300 FREE
apt        lock at @ B0476380 FREE
pdt alloc  lock at @ B0476400 FREE
pdt iolist lock at @ B0476480 FREE
comp       lock at @ B0476500 FREE
zq         lock at @ 006F09C8 FREE
lw         lock at @ 006F08E0 FREE

MEMORY POOLS & FRAMESET LOCKS

VMPOOL 00
        mempool[00000000]: LRU      lock at @ 01FA4004 FREE
                frameset[00000000]: free nfr lock @ 01F94000 FREE
                frameset[00000001]: free nfr lock @ 01F94080 FREE

SCOREBOARD

scoreboard cpu 0 :
hint.....................00000000
```

```
00: empty
01: empty
(0)> more (^C to quit) ?
02: empty
03: empty
04: empty
05: empty
06: empty
07: empty
scoreboard cpu 1 :
hint.....................00000000
00: empty
01: empty
02: empty
03: empty
04: empty
05: empty
06: empty
07: empty
KDB(0)>
```

# vmlog subcommand

## Purpose

The **vmlog** subcommand displays the current VMM error log entry.

## Syntax

**vmlog**

## Parameters

No parameters.

## Aliases

No aliases.

## Example

The following is an example of how to use the **vmlog** subcommand:

```
KDB(0)> vmlog  //display VMM error log entry
Most recent VMM errorlog entry
Error id              =  DSI_PROC
Exception DSISR/ISISR  =  40000000
Exception srval        =  007FFFFF
Exception virt addr    =  FFFFFFFF
Exception value        =  0000000E
KDB(0)> dr iar  //display current instruction
iar   : 01913DF0
01913DF0     lwz    r0,0(r3)            r0=00001030,0(r3)=FFFFFFFF
KDB(0)>
```

# vmpool subcommand

## Purpose

The **vmpool** subcommand displays VMM information for resource pools.

## Syntax

**vmpool** {[**-l** | **-d** | **-f**] ***** | *vmpool_id*}

## Parameters

- **-l** – Indicates that the SYSVMP_LGPG type anchor should be accessed.
- **-d** – Indicates that the SYSVMP_DORM type anchor should be accessed.
- **-f** – Indicates that the SYSVMP_FREE type anchor should be accessed.
- ***** – Indicates that the summary information is to be displayed.
- *vmpool_id* – Indicates the specific vmpool identifier.

The **vmpool** subcommand displays VMM data for resource pools (`struct vmpool_t`). Use the asterisk ( ***** ) parameter to display summary information. The information you select to display can be modified by including one of the flags. If none of the flags are used, the `SYSVMP_NORMAL`-type anchor is accessed.

You can also use the **vmpool** subcommand to display information for a specific vmpool identifier.

## Aliases

No aliases.

## Example

The following is an example of how to use the **vmpool** subcommand:

```
KDB(1)> vmpool *

VMM Resource Pools Data:
VMP NEXT LRUPAGES    MEMPOOLS              FPMP  MEMP_VMINT
00  -1   000026549F  001: 000              002   F100001420000000

KDB(1)> vmpool -l *
No vmpools on this list.

KDB(1)> vmpool -f *

VMM Resource Pools Data:
VMP NEXT
01  02
02  03
03  04
04  05
05  06
06  07
07  08
08  09
09  0A
0A  0B
0B  0C
0C  0D
0D  0E
0E  0F
0F  -1

KDB(1)> vmpool 2
VMPOOL 02 (addr = 000000000027C9B0)
```

```
        number of LRUable pages     (npages_lru)       : 00000000
        sradid                      (srad_id)          : 00000000
        first memory pool           (memp_first)       : FFFFFFFF
        number of memory pools      (nb_mempool)       : 00000000
        number of frame sets / memp (nb_frs_per_memp)  : 00000000
        first nfr on lgpg freelist  (lgpg_free)        : 0000000000000001
        number of frames on lgpg freelist (lgpg_numfrb): 0000000000000000
        total # of lgpg frames      (npages_lg)        : 0000000000000000
        addr of vmintervals array   (vmint)            : 0000000000000000
        addr of freemem list        (freemem)          : 0000000000000000
        addr of usedmem list        (usedmem)          : 0000000000000000
        affinity_list               (affinity_list)    : 000000000027C9F0
                                                        NULL
        next vmpool                 (next)             : 03
        next lgpg vmpool            (next_lgpg)        : 00
        last_[memp/frs]_ecpus                          : 0000 / 0000
        vmpool flags                (flags)            : 00000000
        large page frb lock @ 000000000027CA50 00000000
        memp frs dr    lock @ 000000000027CA58 00000000
KDB(1)>
```

## vmstat subcommand

### Purpose

The **vmstat** subcommand displays virtual memory statistics.

### Syntax

**vmstat**

### Parameters

No parameters.

### Aliases

No aliases.

### Example

The following is an example of how to use the **vmstat** subcommand:

```
KDB(0)> vmstat

VMM Statistics:

page faults              (pgexct)   : 00105695
page reclaims            (pgrclm)   : 00000000
lockmisses               (lockexct) : 00000000
backtracks               (backtrks) : 0000C2A2
pages paged in           (pageins)  : 00004824
pages paged out          (pageouts) : 0000CEFA
paging space page ins    (pgspgins) : 00000000
paging space page outs   (pgspgouts): 00000000
start I/Os               (numsios)  : 0000E251
iodones                  (numiodone): 0000CFAA
zero filled pages        (zerofills): 0007764B
executable filled pages  (exfills)  : 00000E77
pages examined by clock  (scans)    : 00000000
clock hand cycles        (cycles)   : 00000000
page steals              (pgsteals) : 00000000
free frame waits         (freewts)  : 00000000
extend XPT waits         (extendwts): 00000000
pending I/O waits        (pendiowts): 000028C7

VMM Statistics:

total virtual pgs        (numvpages): 000000000000BA03
pages in use for wseg   (numwseguse): 000000000000881F
pages in use for pseg   (numpseguse): 0000000000002D1F
pages in use for clseg (numclseguse): 0000000000001D79
pages pinned for wseg   (numwsegpin): 00000000000037D8
pages pinned for pseg   (numpsegpin): 0000000000000000
pages pinned for clseg (numclsegpin): 0000000000000000
ping-pongs: source => alias (pings) : 00000000
ping-pongs: alias => source (pongs) : 00000000
ping-pongs: alias => alias  (pangs) : 00000000
ping-pongs: alias page del  (dpongs): 00000000
ping-pongs: alias page write(wpongs): 00000000
ping-pong cache flushes     (cachef): 00000000
ping-pong cache invalidates (cachei): 00000000
hardware large page size (lgpg_size): 00000000
total num of large pages  (lgpg_cnt): 0000000000000000
num free large pages   (lgpg_numfrb): 0000000000000000
large page high water cnt  (lgpg_hi): 0000000000000000
large page in-use cnt   (lgpg_inuse): 0000000000000000
```

```
num reserved sids      (numspecsegs): 0000000000000000
num free reserved sids (numspecfree): 0000000000000000
reserved sids hi-water  (specsegshi): 0000000000000000
KDB(0)>
```

## vmthrpgio subcommand

### Purpose

The **vmthrpgio** subcommand provides VMM support of thread/base level page I/O commands.

### Syntax

**vmthrpgio**

### Parameters

No parameters.

When you enter the **vmthrpgio** subcommand, the following options are displayed:

```
1) display a given thrpgio frame structure (user provides the address)
2) display the ut_pgio_fields of the current thread
3) display THRPGIO bufstructs.  The user provides the address of
   a struct bufthrio.  Any av_forw chain is traversed, displaying
   each struct bufthrio.
```

### Aliases

No aliases.

### Example

No example.

## vmwait subcommand

### Purpose

The **vmwait** subcommand displays VMM wait status.

### Syntax

**vmwait** [*effectiveaddress*]

### Parameters

- *effectiveaddress* – Specifies the effective address for a wait channel. Symbols, hexadecimal values, or hexadecimal expressions to specify the address.

If no parameter is used, you are prompted for the wait address.

### Aliases

No aliases.

### Example

The following is an example of how to use the **vmwait** subcommand:

```
KDB(0)> th -w WPGIN
                SLOT NAME     STATE    TID PRI  RQ CPUID  CL WCHAN

pvthread+004600  140 sync     SLEEP 008CF1 03C   1         0 B048CCA0
KDB(0)> vmwait B048CCA0
VMM Wait Info
Waiting on persistent segment I/O level (v_iowait), sidx = 000003CB
KDB(0)>
```

# vrld subcommand

## Purpose

The **vrld** subcommand displays the VMM reload translate table. This information is used only on the SMP POWER-based machine to prevent VMM reload dead-lock.

## Syntax

**vrld**

## Parameters

No parameters.

## Aliases

No aliases.

## Example

The following is an example of how to use the **vrld** subcommand:

```
KDB(0)> vrld

freepno: 0A, initobj: 0008DAA8, *initobj: FFFFFFFF

[00] sid: 00000000, anch: 00
  {00} spno:00000000, epno:00000097, nfr:00000000, next:01
  {01} spno:00000098, epno:000000AB, nfr:00000098, next:02
  {02} spno:FFFFFFFF, epno:000001F6, nfr:000001DD, next:03
  {03} spno:000001F7, epno:000001FA, nfr:000001F7, next:04
  {04} spno:0000038C, epno:000003E3, nfr:00000323, next:FF

[01] sid: 00000041, anch: 06
  {06} spno:00003400, epno:0000341F, nfr:000006EF, next:05
  {05} spno:00003800, epno:00003AFE, nfr:000003F0, next:08
  {08} spno:00006800, epno:00006800, nfr:0000037C, next:07
  {07} spno:00006820, epno:00006820, nfr:0000037B, next:09
  {09} spno:000069C0, epno:000069CC, nfr:0000072F, next:FF

[02] sid: FFFFFFFF, anch: FF

[03] sid: FFFFFFFF, anch: FF

KDB(0)>
```

## vsidd subcommand

### Purpose

The **vsidd** subcommand displays memory using a vsid (virtual segment identifier) and byte offset addressing format.

### Syntax

**vsidd** {*vsid:offset*} [*count*] [**,w**|**,d**]

### Parameters

- *vsid:offset* – Identifies the memory location to be displayed. The *vsid* parameter indicates which segment to access, and the *offset* is the number of bytes into the segment from which to begin displaying. These parameters are required.
- *count* – Indicates the number of display units (4-byte words or 16-byte double words) to display. If count is omitted, one line (32-bytes) of data is displayed.
- **,w** – Indicates that the display unit is 4-byte words.
- **,d** – Indicates that the display unit is 8-byte double words.

**Note:** For the 32-bit kernel, the default display unit is 4 bytes. For the 64-bit kernel, the default display unit is 8 bytes. In both cases, the page must be in memory.

### Aliases

**sidd**

### Example

The following is an example of how to use the **vsidd** subcommand:

```
Display starting at offset 0x80 from the segment containing the IPL control block
(example vsid of 1F0FFF) on the 64-bit kernel:
KDB(0)> vsidd 1F0FFF:80 8
001F0FFF:00000080: 524F5349504C200A 00000000000131F0     ROSIPL .......1.
001F0FFF:00000090: 00000F1C00000007 0000032800000598     ...........(....
001F0FFF:000000A0: 0000000000000000 0000000000000000     ................
001F0FFF:000000B0: 0000000000000000 0000000000000000     ................
KDB(0)> vsidd 1F0FFF:80 8,w
001F0FFF:00000080: 524F5349 504C200A 00000000 000131F0     ROSIPL .......1.
001F0FFF:00000090: 00000F1C 00000007 00000328 00000598     ...........(....
KDB(0)>
```

# vsidm subcommand

## Purpose

The **vsidm** subcommand modifies memory using a vsid (virtual segment identifier) and the byte offset addressing format.

## Syntax

**vsidm** {*vsid:offset*} [**,w**|**,d**]

## Parameters

- *vsid:offset* – Identifies the memory location to be modified. The *vsid* parameter indicates which segment to access, and the *offset* is the first byte to access. These parameters are required.
- **,w** – Indicates that the modification unit is 4-byte words.
- **,d** – Indicates that the modification unit is 8-byte double words.

**Note:** For the 32-bit kernel, the default modification unit is 4 bytes. For the 64-bit kernel, the default modification unit is 8 bytes. In both cases, the page must be in memory.

This **vsidm** subcommand works like other memory-modification commands. The current word (or double word) at the target location is displayed. If you enter a new value, the memory location is changed. If you press Enter without typing a value, the value in the memory location remains unchanged and the next location is displayed for modification. When you type a period (.), the command terminates.

## Aliases

**sidm**

## Example

The following is an example of how to use the **vsidm** subcommand:

```
Modify starting at offset 0x80 from the segment containing the IPL control block
(example vsid of 1F0FFF) on the 64-bit kernel, using word (4 byte) units
KDB(0)> vsidm 1F0FFF:80,w
001F0FFF:00000080:  524F5349  = 4B444249
001F0FFF:00000084:  504C200A  =              <press enter>
001F0FFF:00000088:  00000000  = .
KDB(0)> vsidd 1F0FFF:80,w
001F0FFF:00000080: 4B444249 504C200A 00000000 000131F0     KDBIPL .......1.
KDB(0)> vsidm 1F0FFF:80
001F0FFF:00000080:  4B444249504C200A  = 524F5349504C200A
001F0FFF:00000088:  00000000000131F0  = .
KDB(0)> vsidd 1F0FFF:80,w
001F0FFF:00000080: 524F5349 504C200A 00000000 000131F0     ROSIPL .......1.
KDB(0)>
```

# zproc subcommand

## Purpose

The **zproc** subcommand displays information about the VMM zeroing kproc.

## Syntax

**zproc**

## Parameters

No parameters

## Aliases

No aliases.

## Example

The following is an example of how to use the **zproc** subcommand:

```
KDB(1)> zproc  //display VMM zeroing kproc

VMM zkproc pid = 63CA  tid = 63FB
Current queue info
        Queue resides at 0x0009E3E8 with 10 elements
        Requests    16800    processed    16800     failed        0
        Elements
              sid      pno      npg      pno      npg
        0 - 007FFFFF FFFFFFFF 00000000 FFFFFFFF 00000000
        1 - 007FFFFF FFFFFFFF 00000000 FFFFFFFF 00000000
        2 - 007FFFFF FFFFFFFF 00000000 FFFFFFFF 00000000
        3 - 007FFFFF FFFFFFFF 00000000 FFFFFFFF 00000000
        4 - 007FFFFF FFFFFFFF 00000000 FFFFFFFF 00000000
        5 - 007FFFFF FFFFFFFF 00000000 FFFFFFFF 00000000
        6 - 007FFFFF FFFFFFFF 00000000 FFFFFFFF 00000000
        7 - 007FFFFF FFFFFFFF 00000000 FFFFFFFF 00000000
        8 - 007FFFFF FFFFFFFF 00000000 FFFFFFFF 00000000
        9 - 007FFFFF FFFFFFFF 00000000 FFFFFFFF 00000000
```

# drlist subcommand

## Purpose

The **drlist** subcommand displays VMM data for a **drlist_t** structure.

**Note:** This command is not available on the uni-processor (unix_up) kernel.

## Syntax

**drlist** [*address*]

## Parameters

- *address* – Specifies the memory location to be displayed as a **drlist_t** structure.

The **drlist** command is used to display a **drlist_t** structure. If no parameter is given, the global kernel anchor is examined and the **drlist_t**, (if any), is displayed. If there is no valid outstanding DRlist, a message is displayed.

If the *address* parameter is given, the memory location is displayed as a **drlist_t** structure.

## Aliases

**drl**

## Example

The following is an example of how to use the **drlist** subcommand:

```
KDB(0)> drlist

DRlist @ F10010F01644D700

start frame....... 0000000000270000
end frame......... 0000000000280000
swpfts............ F20080001EA00000
swpfte............ F20080001F000000
pvts.............. F200800021380000
pvte.............. F200800021400000
pftpages.......... 0000000000000000
vmpool_id......... 00000000
memop............. 00000001
flags............. 00000000
freefwd........... 0000000000000001
freebwd........... 0000000000000001
nfree............. 0000000000000000
lruptr............ 0000000003A61430
lruvisits......... 0000000000000000
maxvisits......... 0000000000000000
lrusteals......... 0000000000000000
maxpouts.......... 0000000000000000
lrupouts.......... 0000000000000000
lrupass........... 00000000
addnfr............ 0000000000000000
lock.............. 0000000000000000

KDB(0)>
```

# lrustate subcommand

## Purpose

The **lrustate** displays the lru daemon control variables.

**Note:** These variables reside on the respective lru daemon stack, and only have valid values while the lru daemon is actively running.

## Syntax

**lrustate** [ mempool id ]

## Parameters

- *mempool id* – Is the memory pool identifier that corresponds to the lru daemon whose state you want to examine.

## Aliases

**lru**

## Example

The following is an example of how to use the **lru** alias for the **lrustate** subcommand:

```
KDB(0)> lru -?
lru <mempool id>
KDB(0)> lru 0

LRU State @00B1F520 for mempool 0
*** this is on the MST stack & only valid if fblru running ***

LRU Start nfr        (lru_start)        : 00000000
mempools first nfr   (lru_firstnfr)     : 00000000
numfrb this mempool  (lru_numfrb)       : 00000004
number of steals     (lru_steals)       : 00000000
page goal to steal   (lru_goal)         : 0000001B
npages scanned       (lru_nbscan)       : 00000002
LFBLRU or CFBLRU     (lru_type)         : 00000000 LFBLRU
scans of start nfr (lru_scan_start_cnt) : 00000000
lru revolutions      (lru_rev)          : 00000000
last buckt<bucketsz(lru_small_mem_wrap) : 00000000
fileonly mode        (lru_fileonly)     : 00000000
progress guaranteed  (lru_progress)     : 00000001
fault color          (lru_fault_col)    : 00000173, 371
steal color          (lru_steal_col)    : 00000173, 371
nbuckets scanned     (lru_nbucket)      : 00000001
lru mode             (lru_mode)         : 00000000
wlm regul enabled? (lru_wlm_is_enabled) : 00000001 WLM Regul is ON
request type         (lru_rq)           : 00000009
drbit before pgout   (lru_drbit)        : 00000000
lru_dr running       (lru_dr)           : 00000000
start ccb            (lru_start_ccb)    : 00000000, 0
ccb pass1 left off   (lru_p1_ccb)       : 00000000, 0
current ccb          (lru_cur_ccb)      : 00000000, 0
KDB(0)>
```

# Chapter 25. Address translation subcommands

The subcommands in this category can be used to display address translation information, display and modify **ibat** and **dbat** registers on POWER-based machines, and display and modify Segment Lookaside Buffer (SLB) information. These subcommands include the following:

- tr
- tv
- slb
- mslb
- dbat
- ibat
- mdbat
- mibat

## tr and tv subcommands

### Purpose

The **tr** and **tv** subcommands display address translation information. The **tr** subcommand provides a short format and the **tv** subcommand provides a detailed format.

### Syntax

**tr** *effectiveaddress*

**tv** *effectiveaddress*

### Parameters

- *effectiveaddress* – Specifies the effective address for which translation details are to be displayed. Use symbols, hexadecimal values or hexadecimal expressions to specify the address.

For the **tv** subcommand, all double-hashed entries are dumped when the entry matches the specified effective address. Corresponding physical address and protections are displayed. Page protection ( the **K** bit and the **PP** bits) is displayed according to the current segment and machine state register values.

### Aliases

No aliases.

### Example

The following is an example of how to use the **tr** and the **tv** subcommands:

```
KDB(0)> nm pvthread
Symbol Address : F1000588D0000000
   TOC Address : 01505F20
KDB(0)> tr pvthread
Physical Address = 000000007F964000
KDB(0)> tv pvthread
starting
kdb_get_vsid 1F88D
eaddr F1000588D0000000 sid 000000000001F88D vpage 0000000000000000 hash1 0001F88D
p64pte_cur_addr 0000000002FC4680 sid 000000000001F88D avpi 00 hsel 0 valid 1
rpn 000000000007F964 refbit 1 modbit 1 wimg 2 key 0
____ 000000007F964000 ____ K = 0 PP = 00 ==> read/write

eaddr F1000588D0000000 sid 000000000001F88D vpage 0000000000000000 hash2 00020772
Physical Address = 000000007F964000
KDB(0)>
```

## slb subcommand

### Purpose

The **slb** subcommand displays Segment Lookaside Buffer (SLB) information.

### Syntax

**slb** [**-r**] [*entry*]

### Parameters

- **-r** – Specifies that the current register contents of the SLBs should be displayed. If there are any SLB values, the **slb** subcommand usually displays them for the current context, but does not display the contents of the registers.

  **Note:** This flag is only supported for the KDB kernel debugger.
- *entry* – Specifies the SLB entry to display. If this parameter is not used, all of the SLBs are displayed.

If the underlying hardware platform does not support SLBs, the **slb** subcommand displays a message indicating that the subcommand is unavailable.

### Aliases

No aliases.

### Example

The following is an example of how to use the **slb** subcommand:

```
KDB(0)> slb
00 0000000008000000 0000000000000400 V  01 F000000028000000 0000000021002000 V
02 F000000030000000 00000000013E0400 I  03 FFFFFFFF08000000 00000001C109C080 V
04 FFFFFFFF10000000 00000000D14AD080 I  05 FFFFFFFF20000000 00000001D12FD080 I
06 FFFFFFFF30000000 0000000111131080 I  07 0000000000000000 0000000000000000 I
08 0000000000000000 0000000000000000 I  09 0000000000000000 0000000000000000 I
0A 0000000000000000 0000000000000000 I  0B 0FFFFFFFF8000000 0000000031003C00 V
0C F100009E18000000 00000001E09DE400 V  0D F100008798000000 0000000160876400 V
0E F100008788000000 0000000150875400 V  0F F1000089C8000000 0000000190899400 V
10 F1000000E8000000 00000000B000B400 V  11 F100000BE8000000 00000001B00BB400 V
12 F100000048000000 0000000010001400 V  13 F100000058000000 0000000020002400 V
14 F100009E28000000 00000001F09DF400 V  15 F10000AFB8000000 0000000180AF8400 V
16 F10000AFC8000000 0000000190AF9400 V  17 F10000AFD8000000 00000001A0AFA400 V
18 F200010018000000 0000010010001400 V  19 F200010028000000 0000010020002400 V
1A F200020038000000 0000020030003500 V  1B F100000BE0000000 00000001B00BB400 I
1C F100000040000000 0000000010001400 I  1D F100000050000000 0000000020002400 I
1E F100009C00000000 00000001D09BD400 I  1F F100009E20000000 00000001F09DF400 I
20 F200010010000000 0000010010001400 I  21 F200010020000000 0000010020002400 I
22 F200020030000000 0000020030003500 I  23 F100009D00000000 00000000D09CD400 I
24 F100001420000000 00000001F013F400 I  25 F100009AE0000000 00000000B09AB400 I
26 F10000AFC0000000 0000000190AF9400 I  27 F100001420000000 00000001F013F400 I
28 090000F0D0000000 00000000A09AAC00 I  29 F100009AD0000000 00000000A09AA400 I
2A 090000F030000000 00000001714D7C00 I  2B 090000F040000000 00000001914D9C00 I
2C F10000AFB0000000 0000000180AF8400 I  2D 090000F0F0000000 00000000314C3C00 I
2E 090000F0F0000000 00000000414C4C00 I  2F F10000AFB0000000 0000000180AF8400 I
30 090000F010000000 00000001810F8C00 I  31 090000F020000000 00000000714C7C00 I
32 090000F0D0000000 00000000A09AAC00 I  33 090000F0F0000000 00000000414C4C00 I
34 F10000AFC0000000 0000000190AF9400 I  35 F10000AFD0000000 00000001A0AFA400 I
36 090000F0F0000000 0000000111471C00 I  37 0000000000000000 0000000000000000 I
38 0000000000000000 0000000000000000 I  39 0000000000000000 0000000000000000 I
3A 0000000000000000 0000000000000000 I  3B 0000000000000000 0000000000000000 I
3C 0000000000000000 0000000000000000 I  3D 0000000000000000 0000000000000000 I
3E 0000000000000000 0000000000000000 I  3F 0000000000000000 0000000000000000 I
```

```
KDB(0)> slb 3
03 FFFFFFFF08000000 00000001C109C080 V
 > valid
esid = 0000000FFFFFFFF0
vsid = 00000000001C109C
KsKp = 00 NLC = 001
KDB(0)>
```

## mslb subcommand

### Purpose

The **mslb** subcommand modifies (Segment Lookaside Buffer) SLB information.

### Syntax

**mslb** [**-r**] [*entry*]

### Parameters

- **-r** – Specifies that the current register contents of the SLB should be modified. If the **-r** flag is not used, the **mslb** subcommand changes the SLB value for the current context.

  **Note:** The **-r** flag is only supported for the KDB kernel debugger.
- *entry* – Indicates the specific SLB entry to modify. This value is a decimal value. If no *entry* parameter is provided, the subcommand defaults to entry number 0.

  The update procedure is identical to other modification subcommands. The current value is displayed, and:
  - The value can be altered.
  - The value can be left unmodified if you press Enter. Pressing Enter causes the next SLB to be displayed. The next SLB is displayed only if no *entry* parameter is entered. If you modify a specific SLB entry, the subcommand terminates after it advances past the virtual segment identifier (VSID) double word.
  - The **mslb** subcommand can be terminated if you enter a period (.).

The SLB is treated as two 8-byte double words, referred to as the effective segment identifier (ESID) and the virtual segment identifier (VSID) respectively. If the underlying hardware platform does not support SLBs, the **mslb** subcommand displays a message indicating that the subcommand is unavailable.

### Aliases

No aliases.

### Example

The following is an example of how to use the **mslb** subcommand:

```
KDB(1)> slb 3
03 0000000000000000 000000FFFFFFF000 I
esid = 0000000000000000
vsid = 000000000FFFFFFF
KsKp = 00 NLC = 000
KDB(1)> mslb 3
03 0000000000000000 000000FFFFFFF000 I Entry ESID = FFFFFFFF08000000 <entered new value FFFFFFFF08000000>
03 FFFFFFFF08000000 000000FFFFFFF000 V Entry VSID = 00000001C109C080 <entered new value 00000001C109C080>
KDB(1)> slb 3
03 FFFFFFFF08000000 00000001C109C080 V
 > valid
esid = 0000000FFFFFFFF0
vsid = 00000000001C109C
KsKp = 00 NLC = 001
KDB(1)>
```

# dbat subcommand

## Purpose

On POWER-based machines that implement the block address translation facility, the **dbat** subcommand displays **dbat** registers.

## Syntax

**dbat** [*index*]

## Parameters

- *index* – Specifies the **dbat** register to display. Valid values are `0` through `3`. If no parameter is specified all **dbat** registers are displayed.

## Aliases

No aliases.

## Example

The following is an example of how to use the **dbat** subcommand:

```
KDB(0)> dbat
DBAT0 0000000040001FFE 00000000C000003A
 bepi 000000002000 brpn 000000006000 bl 07FF vs 1 vp 0 wimg 7 pp 2
 eaddr = 0000000040000000, paddr = 00000000C0000000 size = 262144 KBytes [Supervisor state]
DBAT1 0000000050001FFE 00000000C000003A
 bepi 000000002800 brpn 000000006000 bl 07FF vs 1 vp 0 wimg 7 pp 2
 eaddr = 0000000050000000, paddr = 00000000C0000000 size = 262144 KBytes [Supervisor state]
DBAT2 0000000000000000 0000000000000002
 bepi 000000000000 brpn 000000000000 bl 0000 vs 0 vp 0 wimg 0 pp 2
DBAT3 0000000000000000 0000000000000000
 bepi 000000000000 brpn 000000000000 bl 0000 vs 0 vp 0 wimg 0 pp 0
KDB(0)> dbat 0
DBAT0 0000000040001FFE 00000000C000003A
 bepi 000000002000 brpn 000000006000 bl 07FF vs 1 vp 0 wimg 7 pp 2
 eaddr = 0000000040000000, paddr = 00000000C0000000 size = 262144 KBytes [Supervisor state]
```

## ibat subcommand

### Purpose

On POWER-based machines that implement the block address translation facility, the **ibat** subcommand can be used to display ibat registers.

### Syntax

**ibat** [*index*]

### Parameters

- *index* – Specifies the ibat register to display. Valid values are 0 through 3. If no parameter is specified, all **ibat** registers are displayed.

### Aliases

No aliases.

### Example

The following is an example of how to use the **ibat** subcommand:

```
KDB(0)> ibat 0
IBAT0 0000000000000000 0000000000000000
 bepi 000000000000 brpn 000000000000 bl 0000 vs 0 vp 0 wimg 0 pp 0
KDB(0)>
```

# mdbat subcommand

## Purpose

The **mdbat** subcommand is used to modify the **dbat** register. The processor data **bat** register is modified immediately. The word containing the valid bit is set last.

## Syntax

**mdbat** [*index*]

## Parameters

- *index* – Specifies the **dbat** register to modify. Valid values are `0` through `3`.

If no parameter is entered, you are prompted for the values for all **dbat** registers. If a parameter is specified for the **mdbat** subcommand, you are only prompted for the new values for the specified **dbat** register.

You can input both the upper and lower values for each **dbat** register or you can press `Enter` for these values. If the upper and lower values for the register are not entered, the user is prompted for the values for the individual fields of the **dbat** register. To stop entering values, you type a period (.) and press Enter at any prompt.

## Aliases

No aliases.

## Example

The following is an example of how to use the **mdbat** subcommand on a PowerPC 604 RISC Microprocessor:

```
KDB(0)> mdbat 2  //alter bat register 2
BAT register, enter <RC> twice to select BAT field, enter <.> to quit
DBAT2 upper 00000000 =
DBAT2 lower 00000000 =
BAT field, enter <RC> to select field, enter <.> to quit
DBAT2.bepi: 00000000 = 00007FE0
DBAT2.brpn: 00000000 = 00007FE0
DBAT2.bl  : 00000000 = 0000001F
DBAT2.vs  : 00000000 = 00000001
DBAT2.vp  : 00000000 = <CR/LF>
DBAT2.wimg: 00000000 = 00000003
DBAT2.pp  : 00000000 = 00000002
DBAT2 FFC0007E FFC0001A
 bepi 7FE0 brpn 7FE0 bl 001F vs 1 vp 0 wimg 3 pp 2
 eaddr = FFC00000, paddr = FFC00000 size = 4096 KBytes [Supervisor state]
KDB(0)> mdbat 2  //clear bat register 2
BAT register, enter <RC> twice to select BAT field, enter <.> to quit
DBAT2 upper FFC0007E = 0
DBAT2 lower FFC0001A = 0
DBAT2 00000000 00000000
 bepi 0000 brpn 0000 bl 0000 vs 0 vp 0 wimg 0 pp 0
```

# mibat subcommand

## Purpose

The **mibat** subcommand is used to modify the **ibat** register. The processor instruction **bat** register is changed immediately.

## Syntax

**mibat** [*index*]

## Parameters

- *index* – Specifies the **ibat** register to modify. Valid values are 0 through 3.

If no parameter is specified, you are prompted for the values for all **ibat** registers. If a parameter is specified for the **mibat** subcommand, you are only prompted for the new values for the specified **ibat** register.

Input both the upper and lower values for each **ibat** register or press Enter to use these values. If the upper and lower values for the register are not entered, you are prompted for the values for the individual fields of the **ibat** register. You can stop entering values by typing a period (.) at any prompt and pressing Enter.

## Aliases

No aliases.

## Example

The following is an example of how to use the **mibat** subcommand on a PowerPC 604 RISC Microprocessor:

```
KDB(0)> mibat 2
BAT register, enter <RC> twice to select BAT field, enter <.> to quit
IBAT2 upper 00000000 = <CR/LF>
IBAT2 lower 00000000 = <CR/LF>
BAT field, enter <RC> to select field, enter <.> to quit
IBAT2.bepi: 00000000 = <CR/LF>
IBAT2.brpn: 00000000 = <CR/LF>
IBAT2.bl  : 00000000 = 3ff
IBAT2.vs  : 00000000 = 1
IBAT2.vp  : 00000000 = <CR/LF>
IBAT2.wimg: 00000000 = 2
IBAT2.pp  : 00000000 = 2
IBAT2 00000FFE 00000012
 bepi 0000 brpn 0000 bl 03FF vs 1 vp 0 wimg 2 pp 2
 eaddr = 00000000, paddr = 00000000 size = 131072 KBytes [Supervisor state]
```

# Chapter 26. Loader subcommands

The subcommands in this category display the kernel loader entries, add symbols from loaded kernel extensions to the KDB kernel debugger's symbol name cache, and display or remove symbol tables. These subcommands include the following:

- lke
- stbl
- rmst
- lle
- exp

# lke, stbl, and rmst subcommand

## Purpose

The **lke** subcommand displays the kernel loader entries and adds symbols from loaded kernel extensions to the symbol name cache that is used for debugging. The **stbl** subcommand displays the symbol tables. The **rmst** subcommand removes a symbol table.

## Syntax

**lke** [**-l**] [**-l32**] [**-l64**] [**-p** *pslot*] [**-n** *name*] [[**-s**] {*entry* | *effectiveaddress*}] [**-a** *ldr_address*]

**stbl** [*sym_slot* | *ldr_address*]

**rmst** [*sym_slot* | *ldr_address*]

## Parameters

- **-l** – Lists the current entries in the name list cache.
- **-l32** – Displays loader entries for 32-bit shared libraries.
- **-l64** – Displays loader entries for 64-bit shared libraries.
- **-p** *pslot* – Displays the shared-library loader entries for the process slot indicated. The value for *pslot* must be a decimal process slot number.
- **-n** *name* – Displays the loader entry specified by *name*.
- **-s** – Does not display symbols when populating the cache.
- *entry* – Specifies a loader entry. The *entry* parameter must be a decimal value. The specified entry is displayed, and the name list cache is loaded with data for that entry.
- *effectiveaddress* – Specifies an effective address in the text or data area for a loader entry. The specified entry is displayed and the name list cache is loaded with data for that entry. This address can be a hexadecimal value, a symbol, or a hexadecimal expression.
- **-a** *ldr_address* – Displays the loader entry at the specified address, and loads the name list cache with data for that entry. This address can be a hexadecimal value, a symbol, or a hexadecimal expression.
- *sym_slot* – Specifies the slot number. This value must be a decimal number.
- *ldr_address* – Specifies the address of a loader entry. The address can be a hexadecimal value, a symbol, or a hexadecimal expression.

During boot phase, KDB kernel debugger is called to load extension symbol tables. When KDB kernel debugger is called, a message is displayed.

The symbol tables that are available to KDB kernel debugger can be listed with the **stbl** subcommand. If this subcommand is invoked without parameters, a summary of all symbol tables is displayed. Details about a particular symbol table can be obtained by supplying a slot number or the effective address of the loader entry to the **stbl** subcommand.

A symbol table can be removed from KDB kernel debugger using the **rmst** subcommand. This subcommand requires that either a slot number or the effective address for the loader entry of the symbol table be specified.

A symbol name cache is managed inside KDB kernel debugger. The cache is filled with function names with the **lke** [**-s**] { *entry* | *address*} subcommand and the **lke -a** *ldr_address* subcommand. When this cache is full, old entries are replaced by new entries.

If the **lke** subcommand is invoked without parameters, a summary of the kernel loader entries is displayed. The **lke** subcommand parameters **-l32** and **-l64** can be used to list the loader entries for 32-bit and 64-bit shared libraries, respectively. Details can be viewed for individual loader entries by specifying the following:

- Entry number
- Address of the loader entry with the **-a** flag
- Address within the text or data area for a loader entry

The name lists contained in the name list cache area can be reviewed by using the **-l** option.

## Aliases

No aliases.

## Example

The following is an example of how to use the **stbl**, **rmst** and **lke**subcommand when /unix and one driver have symbol tables:

**Note:** If the kernel extension is stripped, the symbol table is not loaded in memory.

```
...//during boot phase
   no symbol [/etc/drivers/mddtu_load]
   no symbol [/etc/drivers/fd]
   Preserving 14280 bytes of symbol table [/etc/drivers/rsdd]
   no symbol [/etc/drivers/posixdd]
   no symbol [/etc/drivers/dtropendd]
   ...
   KDB(4)> stbl //list symbol table entries
       LDRENTRY     TEXT     DATA      TOC MODULE NAME
     1 00000000 00000000 00000000 00207EF0 /unix
     2 0B04C400 0156F0F0 015784F0 01578840 /etc/drivers/rsdd
   KDB(4)> rmst 2  //ignore second entry
   KDB(4)> stbl //list symbol table entries
       LDRENTRY     TEXT     DATA      TOC MODULE NAME
     1 00000000 00000000 00000000 00207EF0 /unix
   KDB(4)> stbl 1 //list a symbol table entry
       LDRENTRY     TEXT     DATA      TOC MODULE NAME
     1 00000000 00000000 00000000 00207EF0 /unix
   st_desc addr.... 00153920
   symoff.......... 002A9EB8
   nb_sym.......... 0000551E

KDB(0)> lke  //summary of kernel loader entries
    ADDRESS     FILE FILESIZE    FLAGS         MODULE NAME

  1 070E6000 03634EA0 0000ADF8 00080272 random64/usr/lib/drivers/random
  2 070DE100 070E1000 00000FF8 00180248 /unix
  3 070E6E00 07541000 00081DC0 00080272 nfs.ext64/usr/lib/drivers/nfs.ext
  4 070E6F00 070DF000 00000FF8 00180248 /unix
  5 070E6C00 03634A60 00000430 00080272 nfs_kdes_null.ext64/usr/lib/drivers/nfs_kdes.ext
  6 070E6D00 07016000 00000FD0 00180248 /unix
  7 070E6B00 036346C0 00000390 00080262 syscalls64.ext64/usr/lib/drivers/syscalls64.ext
  8 070E6900 0362EA60 00005C50 00080272 perfstat64/usr/lib/perf/perfstat
  9 070E6A00 070EE000 00000FD0 00380248 /unix
 10 070E6700 0362E7A0 000002A0 00080262 smt_loadpin64/usr/lib/drivers/smt_loadpin
 11 070E6600 03629DE0 000049A8 00080272 smt_load64/usr/lib/drivers/smt_load
 12 070E6800 070EC000 00000E40 00180248 /unix
 13 070E6400 03616E80 00012F48 00080272 ptydd64/usr/lib/drivers/ptydd
 14 070E6500 070E8000 00000DC0 00180248 /unix
(0)> more (^C to quit) ? ^C  //interrupt
KDB(0)> lke 7  //show loader entry, populate cache
    ADDRESS     FILE FILESIZE    FLAGS         MODULE NAME
```

```
  7 070E6B00 036346C0 00000390 00080262 syscalls64.ext64/usr/lib/drivers/syscall
s64.ext
le_flags....... TEXT DATAINTEXT DATA DATAEXISTS 64
le_next........ 070E6900 le_svc_sequence 00000000
le_fp.......... 00000000
le_filename.... 070E6B88 le_file........ 036346C0
le_filesize.... 00000390 le_data........ 036349B8
le_tid......... 036349B8 le_datasize.... 00000098
le_usecount.... 00000002 le_loadcount... 00000002
le_ndepend..... 00000001 le_maxdepend... 00000001
le_ule......... 00000000 le_deferred.... 00000000
le_exports..... 00000000 le_de.......... 00000000
le_searchlist.. 00000000 le_dlusecount.. 00000000
le_dlindex..... FFFFFFFF le_lex......... 00000000
le_fh.......... 00000000 le_depend.... @ 070E6B80
TOC@........... 03634A28
                             <PROCESS TRACE BACKS>                   .config64 03634870
               .xmalloc.glink 03634940
               .copyin.glink 03634968          .ldr_config64.glink 03634990
KDB(0)> lke -s 7  //show loader entry, populate cache without printing symbols
    ADDRESS     FILE FILESIZE    FLAGS         MODULE NAME

  7 070E6B00 036346C0 00000390 00080262 syscalls64.ext64/usr/lib/drivers/syscall
s64.ext
le_flags....... TEXT DATAINTEXT DATA DATAEXISTS 64
le_next........ 070E6900 le_svc_sequence 00000000
le_fp.......... 00000000
le_filename.... 070E6B88 le_file........ 036346C0
le_filesize.... 00000390 le_data........ 036349B8
le_tid......... 036349B8 le_datasize.... 00000098
le_usecount.... 00000002 le_loadcount... 00000002
le_ndepend..... 00000001 le_maxdepend... 00000001
le_ule......... 00000000 le_deferred.... 00000000
le_exports..... 00000000 le_de.......... 00000000
le_searchlist.. 00000000 le_dlusecount.. 00000000
le_dlindex..... FFFFFFFF le_lex......... 00000000
le_fh.......... 00000000 le_depend.... @ 070E6B80
TOC@........... 03634A28
KDB(0)> lke -a 070E6B00  //show loader entry by address, populate cache
    ADDRESS     FILE FILESIZE    FLAGS         MODULE NAME

    070E6B00 036346C0 00000390 00080262 syscalls64.ext64/usr/lib/drivers/syscall
s64.ext
le_flags....... TEXT DATAINTEXT DATA DATAEXISTS 64
le_next........ 070E6900 le_svc_sequence 00000000
le_fp.......... 00000000
le_filename.... 070E6B88 le_file........ 036346C0
le_filesize.... 00000390 le_data........ 036349B8
le_tid......... 036349B8 le_datasize.... 00000098
le_usecount.... 00000002 le_loadcount... 00000002
le_ndepend..... 00000001 le_maxdepend... 00000001
le_ule......... 00000000 le_deferred.... 00000000
le_exports..... 00000000 le_de.......... 00000000
le_searchlist.. 00000000 le_dlusecount.. 00000000
le_dlindex..... FFFFFFFF le_lex......... 00000000
le_fh.......... 00000000 le_depend.... @ 070E6B80
TOC@........... 03634A28
                             <PROCESS TRACE BACKS>
                   .config64 03634870              .xmalloc.glink 03634940
               .copyin.glink 03634968          .ldr_config64.glink 03634990
KDB(0)> lke -l  //list the cache
                             KERNEXT FUNCTION NAME CACHE
                   .config64 03634870             .xmalloc.glink 03634940
               .copyin.glink 03634968         .ldr_config64.glink 03634990
00 KERNEXT FUNCTION range [03634870 036349A8] 4 entries
KDB(0)> lke -l32  //loader entries for 32-bit shared libraries
    ADDRESS     FILE FILESIZE    FLAGS         MODULE NAME
```

```
  1 F100009AE00E8600 D0CDE000 0000491C 00000882 /usr/lib/nls/loc/uconv/UTF32TBL
  2 F100009AE00E8500 D017E000 00002663 00000882 /usr/lib/nls/loc/iconv/UTF-32_UTF-8
  3 F100009AE00E8400 D0CCF0C0 0000E73A 000000C0 shr.o/usr/lib/libct_di.a
  4 F100009AE00E8300 D0CC70C0 00006FB2 000000C0 shr.o/usr/lib/libcsm_clog.a
  5 F100009AE00E8200 D0CCF0C0 0000E73A 00000882 shr.o/usr/lib/libct_di.a
  6 F100009AE00E8100 D0CC70C0 00006FB2 00000882 shr.o/usr/lib/libcsm_clog.a
  7 F100009AE00CE000 D0BEF0C0 000D706B 000000C0 shr.o/usr/lib/libct_mc.a
  8 F100009AE00CEF00 D0BEF0C0 000D706B 00000882 shr.o/usr/lib/libct_mc.a
  9 F100009AE00CED00 D0BB50C0 00039B41 000000C0 shr.o/usr/lib/libct_sr.a
 10 F100009AE00CEC00 D0B4E0C0 0006666F 000000C0 shr.o/usr/lib/libct_rm.a
 11 F100009AE00CEB00 D0A3E0C0 0010FDEE 000000C0 shr.o/usr/lib/libct_rmf.a
 12 F100009AE00CEA00 D0A0A0C0 00033A77 000000C0 shr.o/usr/lib/libct_dev.a
 13 F100009AE00CE900 D0BB50C0 00039B41 00000882 shr.o/usr/lib/libct_sr.a
 14 F100009AE00CE800 D0B4E0C0 0006666F 00000882 shr.o/usr/lib/libct_rm.a
(0)> more (^C to quit) ? ^C  //interrupt
KDB(0)> lke -l64  //loader entries for 64-bit shared libraries
    ADDRESS     FILE FILESIZE    FLAGS         MODULE NAME

  1 F100009F30049F00 900000000051AC0 0001073D 000800C0 shr_64.o/usr/lib/libcfg.a
  2 F100009F30049E00 900000000045920 0000A898 000800C0 shr_64.o/usr/lib/libdpi20.a
  3 F100009F30049D00 9000000000319C0 000133D1 000800C0 shr_64.o/usr/lib/libsrc.a
  4 F100009F30049C00 90000000001C360 0001488C 000800C0 shr_64.o/usr/lib/libodm.a
  5 F100009F30049B00 900000000063280 00000A2B 000800C0 shr_64.o/usr/lib/libcrypt.a
  6 F100009F30049A00 900000000243000 00223526 000800C0 shr_64.o/usr/lib/libc.a
  7 F100009F30049900 900000000063280 00000A2B 00080882 shr_64.o/usr/lib/libcrypt.a
  8 F100009F30049800 900000000051AC0 0001073D 00080882 shr_64.o/usr/lib/libcfg.a
  9 F100009F30049700 900000000045920 0000A898 00080882 shr_64.o/usr/lib/libdpi20.a
 10 F100009F30049600 9000000000319C0 000133D1 00080882 shr_64.o/usr/lib/libsrc.a
 1 1 F100009F30049400 90000000001C360 0001488C 00080882 shr_64.o/usr/lib/libodm.a
 12 F100009F30049500 900000000243000 00223526 00080882 shr_64.o/usr/lib/libc.a
KDB(0)> lke -p 1  //loader entries for process slot 1
    ADDRESS     FILE FILESIZE    FLAGS         MODULE NAME

  1 F00000002FFC8300 D004E000 0002BAEB 00021740 shr_xpg5.o/usr/lib/libpthreads.a
  2 F00000002FFC8200 D004A000 000038C7 00001740 shr_comm.o/usr/lib/libpthreads.a
  3 F00000002FFC8100 D007A0F8 00000846 00001740 shr.o/usr/lib/libcrypt.a
  4 F00000002FF3C578 D01DEE00 001F800B 00001740 shr.o/usr/lib/libc.a
  5 F00000002FF3C4C0 10000000 0000850E 00005242 init
KDB(0)> lke -n syscalls64.ext64  //loader entry by name
    ADDRESS     FILE FILESIZE    FLAGS         MODULE NAME

  7 070E6B00 036346C0 00000390 00080262 syscalls64.ext64/usr/lib/drivers/syscalls64.ext
```

## lle subcommand

### Purpose

The **lle** subcommand lists loader entries.

### Syntax

**lle** [**-k** | **-l32** | **-l64** | **-a** *addr*] [**-p** *slot*] [**-A**] [**-v**]

### Parameters

- **-k** – Lists the kernel loader entries.
- **-l32** – Lists the 32-bit library loader entries.
- **-l64** – Lists the 64-bit library loader entries.
- **-a** – Lists the loader entry at the specified address.
- **-p** – Lists the loader entries for the specified process.
- **-A** – Lists the loader anchor information.
- **-v** – Lists all fields in the selected entries.
- *address* – Specifies the address of a loader entry. Symbols, hexadecimal values, or hexadecimal expressions can be used to specify the address.
- *slot* - Specifies a decimal process slot.

### Aliases

No aliases.

### Example

The following is an example of how to use the **lle** subcommand:

```
KDB(0)> lle -k                          //kernel loader entries
    ADDRESS     FILE FILESIZE    FLAGS         MODULE NAME

  1 07058000 03634EA0 0000ADF8 00080272  /usr/lib/drivers/random(random64)
  2 07172100 07175000 00000FF8 00180248  /unix
  3 07058E00 07541000 00081DC0 00080272  /usr/lib/drivers/nfs.ext(nfs.ext64)
  4 07058F00 07173000 00000FF8 00180248  /unix
  5 07058C00 03634A60 00000430 00080272  /usr/lib/drivers/nfs_kdes.ext(nfs_kdes_null.ext64)
  6 07058D00 07170000 00000FD0 00180248  /unix
  7 07058B00 036346C0 00000390 00080262  /usr/lib/drivers/syscalls64.ext(syscalls64.ext64)
  8 07058900 0362EA60 00005C50 00080272  /usr/lib/perf/perfstat(perfstat64)
  9 07058A00 0717A000 00000FD0 00380248  /unix
 10 07058700 0362E7A0 000002A0 00080262  /usr/lib/drivers/smt_loadpin(smt_loadpin64)
 11 07058600 03629DE0 000049A8 00080272  /usr/lib/drivers/smt_load(smt_load64)
 12 07058800 07178000 00000E40 00180248  /unix
 13 07058400 03616E80 00012F48 00080272  /usr/lib/drivers/ptydd(ptydd64)
 14 07058500 0716E000 00000DC0 00180248  /unix
 15 07058300 035FC940 0001A518 00080262  /usr/lib/drivers/iscsidd(iscsidd64)
 16 07058100 035F80E0 00004838 00080272  /usr/lib/drivers/if_en(if_en64)
 17 07058200 07016000 00000DB8 00180248  /unix
 18 07013F00 072AC000 001265F0 00080272  /usr/lib/drivers/netinet(netinet64)
 19 07013000 07017000 00000DB8 01180248  /unix
 20 07013E00 035F1E20 000062A0 00080262  /usr/lib/drivers/isa/msedd_chrp(msedd_chrp64)
(0)> more (^C to quit) ? ^C              //interrupt
KDB(0)> lle -l32                    //32-bit library loader entries
    ADDRESS     FILE FILESIZE    FLAGS         MODULE NAME

  1 F100009AE00DA000 D0D1F0F8 00004597 000000C0  /usr/lib/libc128.a(shr.o)
  2 F100009AE00DAF00 D0E43F60 00006807 000000C0  /usr/lib/libC128.a(shr3.o)
  3 F100009AE00DAE00 D0D24C80 0011D5C6 000000C0  /usr/lib/libC128.a(ansi_32.o)
  4 F100009AE00DAD00 D0CEC100 0003258B 000000C0  /usr/lib/libC128.a(shr.o)
```

```
  5 F100009AE00DAC00 D0CE31A0 00007EF4 000000C0  /usr/lib/libC128.a(shr2.o)
  6 F100009AE00DAB00 D0E43F60 00006807 00000882  /usr/lib/libC128.a(shr3.o)
  7 F100009AE00DAA00 D0D24C80 0011D5C6 00000882  /usr/lib/libC128.a(ansi_32.o)
  8 F100009AE00DA900 D0D1F0F8 00004597 00000882  /usr/lib/libc128.a(shr.o)
  9 F100009AE00DA800 D0CEC100 0003258B 00000882  /usr/lib/libC128.a(shr.o)
 10 F100009AE00DA700 D0CE31A0 00007EF4 00000882  /usr/lib/libC128.a(shr2.o)
 11 F100009AE00DA600 D0CDE000 0000491C 00000882  /usr/lib/nls/loc/uconv/UTF32TBL
 12 F100009AE00DA500 D017E000 00002663 00000882  /usr/lib/nls/loc/iconv/UTF-32_UTF-8
 13 F100009AE00DA400 D0C950C0 0000E73A 000000C0  /usr/lib/libct_di.a(shr.o)
 14 F100009AE00DA300 D0BBD0C0 000D706B 000000C0  /usr/lib/libct_mc.a(shr.o)
 15 F100009AE00DA200 D0B4E0C0 00006FB2 000000C0  /usr/lib/libcsm_clog.a(shr.o)
 16 F100009AE00CE000 D0CA40C0 00039B41 000000C0  /usr/lib/libct_sr.a(shr.o)
 17 F100009AE00CEF00 D0B560C0 0006666F 000000C0  /usr/lib/libct_rm.a(shr.o)
 18 F100009AE00CEE00 D0A3E0C0 0010FDEE 000000C0  /usr/lib/libct_rmf.a(shr.o)
 19 F100009AE00CED00 D0A0A0C0 00033A77 000000C0  /usr/lib/libct_dev.a(shr.o)
 20 F100009AE00CEC00 D0CA40C0 00039B41 00000882  /usr/lib/libct_sr.a(shr.o)
(0)> more (^C to quit) ? ^C               //interrupt
KDB(0)> lle -l64                    //64-bit library loader entries
    ADDRESS     FILE FILESIZE    FLAGS         MODULE NAME

  1 F100009F30049D00 900000000279AC0 0001073D 000800C0  /usr/lib/libcfg.a(shr_64.o)
  2 F100009F30049C00 90000000026D920 0000A898 000800C0  /usr/lib/libdpi20.a(shr_64.o)
  3 F100009F30049B00 9000000002599C0 000133D1 000800C0  /usr/lib/libsrc.a(shr_64.o)
  4 F100009F30049A00 900000000244360 0001488C 000800C0  /usr/lib/libodm.a(shr_64.o)
  5 F100009F30049900 90000000028B280 00000A2B 000800C0  /usr/lib/libcrypt.a(shr_64.o)
  6 F100009F30049800 900000000020000 00223526 000800C0  /usr/lib/libc.a(shr_64.o)
  7 F100009F30049700 90000000028B280 00000A2B 00080882  /usr/lib/libcrypt.a(shr_64.o)
  8 F100009F30049600 900000000279AC0 0001073D 00080882  /usr/lib/libcfg.a(shr_64.o)
  9 F100009F30049500 90000000026D920 0000A898 00080882  /usr/lib/libdpi20.a(shr_64.o)
 10 F100009F30049400 9000000002599C0 000133D1 00080882  /usr/lib/libsrc.a(shr_64.o)
 11 F100009F30049300 900000000244360 0001488C 00080882  /usr/lib/libodm.a(shr_64.o)
 12 F100009F30049200 900000000020000 00223526 00080882  /usr/lib/libc.a(shr_64.o)
(0)> more (^C to quit) ? ^C               //interrupt
KDB(0)> lle -a 07058000                   //loader entry at a specific address
    Loader Entry @07058000
  le_filename.... 07058088 /usr/lib/drivers/random(random64)
  le_flags....... TEXT KERNELEX DATAINTEXT DATA DATAEXISTS 64
  le_next........ 07172100   le_svc_sequence 00FFFFFF
  le_fp.......... 00000000
  le_fh.......... 00000000   le_file........ 03634EA0
  le_filesize.... 0000ADF8   le_data........ 0363AC80
  le_tid......... 0363AC80   le_datasize.... 00005018
  le_usecount.... 00000003   le_loadcount... 00000001
  le_ndepend..... 00000001   le_maxdepend... 00000001
  le_deferred.... 00000000   le_ule......... 00000000
  le_exports..... 07601000   le_de.......... F00E000000000002
  le_searchlist.. 00000000   le_dlusecount.. 00000000
  le_dlindex..... FFFFFFFF   le_lex......... 00000000
  le_depend...... 07058E00
KDB(0)> lle -p 1                    //loader entries for process slot 1
    ADDRESS     FILE FILESIZE    FLAGS         MODULE NAME

  1 F00000002FFC8300 D004E000 0002BAEB 00021740  /usr/lib/libpthreads.a(shr_xpg5.o)
  2 F00000002FFC8200 D004A000 000038C7 00001740  /usr/lib/libpthreads.a(shr_comm.o)
  3 F00000002FFC8100 D007A0F8 00000846 00001740  /usr/lib/libcrypt.a(shr.o)
  4 F00000002FF3C578 D01DEE00 001F800B 00001740  /usr/lib/libc.a(shr.o)
  5 F00000002FF3C4C0 10000000 0000850E 00005242  init
KDB(0)> lle -p 1 -v                 //verbose output
  1 Loader Entry @F00000002FFC8300
  le_filename.... F100009AE0049588 /usr/lib/libpthreads.a(shr_xpg5.o)
  le_flags....... DATA LIBEXPORTS DATAEXISTS USEASIS DATAMAPPED RTINIT_SEEN
  le_next........ F00000002FFC8200   le_svc_sequence 00000000
  le_fp.......... F100009D00004FD0
  le_fh.......... F10000F0052FD428   le_file........ D004E000
  le_filesize.... 0002BAEB   le_data........ F0123000
  le_tid......... F0123000   le_datasize.... 0000500C
  le_usecount.... 00000002   le_loadcount... 00000000
```

```
  le_ndepend..... 00000004   le_maxdepend... 00000004
  le_deferred.... 00000000   le_ule......... 00000000
  le_exports..... F100009AE0067000   le_de.......... 00000000
  le_searchlist.. F00000002FFCA080   le_dlusecount.. 00000000
  le_dlindex..... 00000003   le_lex......... 00000000
  le_depend...... F100009AE0049600
                  F00000002FFC8200 /usr/lib/libpthreads.a(shr_comm.o)
                  F00000002FF3C578 /usr/lib/libc.a(shr.o)
                  0701AD00 /unix

  2 Loader Entry @F00000002FFC8200
  le_filename.... F100009AE0049788 /usr/lib/libpthreads.a(shr_comm.o)
  le_flags....... DATA LIBEXPORTS DATAEXISTS USEASIS DATAMAPPED
(0)> more (^C to quit) ? ^C               //interrupt
KDB(0)> lle -p 1 -A                    //loader anchor information
ANCHOR ADDRESS... F00000002FF3C400
la_loadlist...... F00000002FFC8300
la_flags......... DEFERRED DATA_HEAP
la_lib_le_sid.... 0000B9AB
ldr64............ 00D05160
```

# exp subcommand

## Purpose

The **exp** subcommand looks for an exported symbol or displays the entire export list.

## Syntax

**exp** [*symbol*]

## Parameters

- *symbol* – Specifies the symbol name to locate in the export list. This parameter is an ASCII string.

If no parameter is specified, the entire export list is displayed. If a symbol name is specified as a parameter and that symbol is in the export list, then that symbol name is displayed. If a symbol name is specified that is not in the list, then all symbols that begin with the input string are displayed.

## Aliases

No aliases.

## Example

The following is an example of how to use the **exp** subcommand:

```
   KDB(0)> exp  //list export table
   000814D4 pio_assist
   019A7708 puthere
   0007BE90 vmminfo
   00081FD4 socket
   01A28A50 tcp_input
   01A28BFC in_pcb_hash_del
   019A78E8 adjmsg
   0000BAB8 execexit
   00325138 loif
   01980874 lvm_kp_tid
   000816E4 ns_detach
   019A7930 mps_wakeup
   01A28C50 ip_forward
   00081E60 ksettickd
   000810AC uiomove
   000811EC blkflush
   0018D97C setpriv
   01A5CD38 clntkudp_init
   000820D0 soqremque
   00178824 devtosth
   00081984 rtinithead
   01A5CD8C xdr_rmtcall_args
   (0)> more (^C to quit) ? ^C //interrupt
KDB(0)> exp send //display symbol 'send'
   007EF084 send
KDB(0)> exp sen //display all symbols that start with 'sen'
   ........  2573 export entries
   007EF54C send_file
   007EF078 sendmsg
   007EF090 sendto
   007F5B38 send_file_duration
   007EF084 send
KDB(0)>
```

# Chapter 27. Display context information subcommands

The subcommands in this category display context information. These subcommands include the following:

- pnda
- ppda
- mst
- lastbackt
- ttid
- tpid
- rq
- rqi
- lq
- cr
- svmon
- meml
- cred

 

## pnda subcommand

### Purpose

The **pnda** subcommand displays the per-node data area **pnda** structures for each processor.

### Syntax

**pnda** [ * | **-a** | *cpu* | *effectiveaddress* ]

### Parameters

- * – Displays a summary of the **pnda** structure for each processor. Multiple processors can share the same **pnda** structure.
- **-a** – Causes the subcommand to display the **pnda** structure associated with each processor on the system.
- *cpu* – Specifies the number of the processor for which you want to display the **pnda** structure.
- *effectiveaddress* – Displays the effective address for which you want to display the **pnda** structure.

When used without parameters, the **pnda** subcommand displays the **pnda** structure for the current processor. With parameters, the **pnda** subcommand can either display a summary of all **pnda** structures on the system, or it can display a **pnda** structure for a specific processor.

### Aliases

No aliases.

### Example

The following is an example of how to use the **pnda** subcommand:

```
KDB(0)> pnda *
          CPU SRAD  CPUBITM          MEMPOOL_ON_SRAD MRQ_SRAD    RSET ATT_ENTRY

00566B50    0    0 F000000000000000 0000000000000000 02171000 0040FD50 00566D20
00566B50    1    0 F000000000000000 0000000000000000 02171000 0040FD50 00566D20
00566B50    2    0 F000000000000000 0000000000000000 02171000 0040FD50 00566D20
00566B50    3    0 F000000000000000 0000000000000000 02171000 0040FD50 00566D20
KDB(0)> pnda 00566B50  //pnda address from the first column of previous subcommand
          CPU SRAD  CPUBITM          MEMPOOL_ON_SRAD MRQ_SRAD    RSET ATT_ENTRY

00566B50    0    0 F000000000000000 0000000000000000 02171000 0040FD50 00566D20

sradid..................00000000
pndas[0]................00566B50
cpu2srad[00]................0000  cpu2srad[01]................0000
cpu2srad[02]................0000  cpu2srad[03]................0000
srad2cpu[0].................0000
cpubitm[0]......F000000000000000
num_cpus_onl[0].........00000004
max_cpus[0].............00000004
max_num_srads...........00000001  num_srads_onl...........00000001
sys_cpus_onl............00000004  sys_max_cpus............00000004
first_srad_with_cpus...........00000000
memp_on_srad[0].0000000000000000
mrq_srad................02171000  gc_heap.................00000000
srad_rptr...............0040FD50  srad_rset...............00566D18
srad_att_entry..........00566D20
netkmem.................3287D000
entry.start.....0000000000000000  entry.nbytes............00000000
```

```
entry.next..............00000000  entry.policy............00000000
entry.cursor............00000000  entry.rset..............00566D38
KDB(0)>
```

## ppda subcommand

### Purpose

The **ppda** subcommand displays a summary for all **ppda** structures with the ***** parameter. Otherwise, details for the current or specified processor are displayed.

### Syntax

**ppda** [***** | **cpu** | *effectiveaddress*]

### Parameters

- ***** – Displays a summary for all processors.
- **cpu** – Displays the data for the **ppda** structure for the specified processor. This parameter must be a decimal value.
- *effectiveaddress* – Specifies the effective address of a **ppda** structure to display. Use symbols, hexadecimal values, or hexadecimal expressions to specify the address.

### Aliases

No aliases.

### Example

The following is an example of how to use the **ppda** subcommand:

```
KDB(1)> ppda *
  SLT   CSA     CURTHREAD     SRR1     SRR0              ppda+000000
   0 004ADEB0 thread+000178 4000D030 1002DC74            ppda+000300
   1 004B8EB0 thread+000234 00009030 ld_usecount+00045C ppda+000600
   2 004C3EB0 thread+0002F0 0000D030 D00012F0            ppda+000900
   3 004CEEB0 thread+0003AC 0000D030 D00012F0            ppda+000C00
   4 004D9EB0 thread+000468 0000F030 D00012F0            ppda+000F00
   5 004E4EB0 thread+000524 0000D030 10019870            ppda+001200
   6 004EFEB0 thread+0005E0 0000D030 D00012F0            ppda+001500
   7 004FAEB0 thread+00069C 0000D030 D00012F0
  KDB(1)> ppda //current processor data area

Per Processor Data Area [000C0300]

csa.....................004B8EB0  mstack...................004B7EB0
fpowner.................00000000  curthread................E6000234
syscall.................0001879B  intr.....................E0100080
i_softis....................0000  i_softpri....................4000
prilvl..................05CB1000
ppda_pal[0].............00000000  ppda_pal[1].............00000000
ppda_pal[2].............00000000  ppda_pal[3].............00000000
phy_cpuid...................0001  ppda_fp_cr..............28222881
flih save[0]............00000000  flih save[1]............2FF3B338
flih save[2]............002E65E0  flih save[3]............00000003
flih save[4]............00000002  flih save[5]............00000006
flih save[6]............002E6750  flih save[7]............00000000
dsisr...................40000000  dsi_flag................00000003
dar.....................2FF9F884
dssave[0]...............2FF3B2A0  dssave[1]...............002E65E0
dssave[2]...............00000000  dssave[3]...............002A4B1C
dssave[4]...............E6001ED8  dssave[5]...............00002A33
dssave[6]...............00002A33  dssave[7]...............00000001
dssrr0..................0027D5AC  dssrr1..................00009030
dssprg1.................2FF9F880  dsctr...................00000000
dslr....................0027D4CC  dsxer...................20000000
dsmq....................00000000  pmapstk.................00212C80
pmapsave64..............00000000  pmapcsa.................00000000
```

```
schedtail[0].............00000000  schedtail[1].............00000000
schedtail[2].............00000000  schedtail[3].............00000000
cpuid....................00000001  stackfix.................00000000
lru......................00000000  vmflags..................00010000
sio............................00  reservation....................01
hint...........................00  lock...........................00
no_vwait.................00000000
scoreboard[0]............00000000
scoreboard[1]............00000000
scoreboard[2]............00000000
scoreboard[3]............00000000
scoreboard[4]............00000000
scoreboard[5]............00000000
scoreboard[6]............00000000
scoreboard[7]............00000000
intr_res1................00000000  intr_res2................00000000
mpc_pend.................00000000  iodonelist...............00000000
affinity.................00000000  TB_ref_u.................003DC159
TB_ref_l.................28000000  sec_ref..................33CDD7B0
nsec_ref.................13EF2000  _ficd....................00000000
decompress...............00000000  ppda_qio.................00000000
cs_sync..................00000000
ppda_perfmon_sv[0].......00000000  ppda_perfmon_sv[1].......00000000
thread_private...........00000000  cpu_priv_seg.............60017017
fp flih save[0]..........00000000  fp flih save[1]..........00000000
fp flih save[2]..........00000000  fp flih save[3]..........00000000
fp flih save[4]..........00000000  fp flih save[5]..........00000000
fp flih save[6]..........00000000  fp flih save[7]..........00000000
TIMER....................
t_free...................00000000  t_active.................05CB9080
t_freecnt................00000000  trb_called...............00000000
systimer.................05CB9080  ticks_its................00000051
ref_time.tv_sec..........33CDD7B1  ref_time.tv_nsec.........01DCDA38
time_delta...............00000000  time_adjusted............05CB9080
wtimer.next..............05767068  wtimer.prev..............0B30B81C
wtimer.func..............000F2F0C  wtimer.count.............00000000
wtimer.restart...........00000000  w_called.................00000000
trb_lock.................000C04F0 slock/slockp 00000000
KDB......................
flih_llsave[0]...........00000000  flih_llsave[1]...........2FF22FB8
flih_llsave[2]...........00000000  flih_llsave[3]...........00000000
flih_llsave[4]...........00000000  flih_llsave[5]...........00000000
flih_save[0].............00000000  flih_save[1].............00000000
flih_save[2].............00000000  csa......................001D4800
KDB(3)>
```

## mst subcommand

### Purpose

The **mst** subcommand prints the Machine State Save Area.

### Syntax

**mst** [*slot*] [[**-a**] *effectiveaddress*]

### Parameters

- **-a** *effectiveaddress* – Specifies the effective address of a Machine State Save Area to display. Use symbols, hexadecimal values, or hexadecimal expressions to specify the address.
- *slot* – Specifies the thread slot number. This value must be a decimal value.

If a thread slot number is specified, the Machine State Save Area for the specified slot is displayed. If an effective address is entered, it is assumed to be the address of the Machine State Save Area.

### Aliases

No aliases.

### Example

The following is an example of how to use the **mst** subcommand:

```
KDB(0)> mst  //current mst

Machine State Save Area
iar   : 0002599C  msr   : 00009030  cr    : 20000000  lr    : 000259B8
ctr   : 000258EC  xer   : 00000000  mq    : 00000000
r0  : 00000000  r1  : 2FF3B338  r2  : 002E65E0  r3  : 00000003  r4  : 00000002
r5  : 00000006  r6  : 002E6750  r7  : 00000000  r8  : DEADBEEF  r9  : DEADBEEF
r10 : DEADBEEF  r11 : 00000000  r12 : 00009030  r13 : DEADBEEF  r14 : DEADBEEF
r15 : DEADBEEF  r16 : DEADBEEF  r17 : DEADBEEF  r18 : DEADBEEF  r19 : DEADBEEF
r20 : DEADBEEF  r21 : DEADBEEF  r22 : DEADBEEF  r23 : DEADBEEF  r24 : DEADBEEF
r25 : DEADBEEF  r26 : DEADBEEF  r27 : DEADBEEF  r28 : 000034E0  r29 : 000C6158
r30 : 000C0578  r31 : 00005004
s0  : 00000000  s1  : 007FFFFF  s2  : 0000F00F  s3  : 007FFFFF  s4  : 007FFFFF
s5  : 007FFFFF  s6  : 007FFFFF  s7  : 007FFFFF  s8  : 007FFFFF  s9  : 007FFFFF
s10 : 007FFFFF  s11 : 007FFFFF  s12 : 007FFFFF  s13 : 0000C00C  s14 : 00004004
s15 : 007FFFFF
prev      00000000 kjmpbuf   00000000 stackfix  00000000 intpri    0B
curid     00000306 sralloc   E01E0000 ioalloc   00000000 backt     00
flags     00 tid       00000000 excp_type 00000000
fpscr     00000000 fpeu            00 fpinfo          00 fpscrx    00000000
o_iar     00000000 o_toc     00000000 o_arg1    00000000
excbranch 00000000 o_vaddr   00000000 mstext    00000000
Except :
 csr 2FEC6B78 dsisr 40000000  bit set: DSISR_PFT
 srval 000019DD dar 2FEC6B78 dsirr 00000106
KDB(0)> mst 1  //slot 1 is thread+0000A0

Machine State Save Area
iar   : 00038ED0  msr   : 00001030  cr    : 2A442424  lr    : 00038ED0
ctr   : 002BCC00  xer   : 00000000  mq    : 00000000
r0  : 60017017  r1  : 2FF3B300  r2  : 002E65E0  r3  : 00000000  r4  : 00000002
r5  : E60000BC  r6  : 00000109  r7  : 00000000  r8  : 000C0300  r9  : 00000001
r10 : 2FF3B380  r11 : 00000000  r12 : 00001030  r13 : 00000001  r14 : 2FF22F54
r15 : 2FF22F5C  r16 : DEADBEEF  r17 : DEADBEEF  r18 : 0000040F  r19 : 00000000
r20 : 00000000  r21 : 00000003  r22 : 01000001  r23 : 00000001  r24 : 00000000
r25 : E600014C  r26 : 000D1A08  r27 : 00000000  r28 : E3000160  r29 : E60000BC
r30 : 00000004  r31 : 00000004
```

```
s0  : 00000000  s1  : 007FFFFF  s2  : 0000A00A  s3  : 007FFFFF  s4  : 007FFFFF
s5  : 007FFFFF  s6  : 007FFFFF  s7  : 007FFFFF  s8  : 007FFFFF  s9  : 007FFFFF
s10 : 007FFFFF  s11 : 007FFFFF  s12 : 007FFFFF  s13 : 6001F01F  s14 : 00004004
s15 : 60004024
prev      00000000 kjmpbuf   00000000 stackfix  2FF3B300 intpri    00
curid     00000001 sralloc   E01E0000 ioalloc   00000000 backt     00
flags     00 tid       00000000 excp_type 00000000
fpscr     00000000 fpeu            00 fpinfo          00 fpscrx    00000000
o_iar     00000000 o_toc     00000000 o_arg1    00000000
excbranch 00000000 o_vaddr   00000000 mstext    00000000
Except :
 csr   30002F00 dsisr 40000000  bit set: DSISR_PFT
 srval 6000A00A dar   20022000 dsirr 00000106

KDB(0)> set 11  //64-bit printing mode
64_bit is true
KDB(0)> sw u   //select user context
KDB(0)> mst    //print user context

Machine State Save Area
iar   : 08000001000581D4  msr   : 800000004000D0B0  cr    : 84002222
lr    : 000000010000047C  ctr   : 08000001000581D4  xer   : 00000000
mq    : 00000000  asr   : 0000000013619001
r0  : 08000001000581D4  r1  : 0FFFFFFFFFFFFF00  r2  : 080000018007BC80
r3  : 0000000000000064  r4  : 0000000000989680  r5  : 0000000000000000
r6  : 800000000000D0B0  r7  : 0000000000000000  r8  : 000000002FF9E008
r9  : 0000000013619001  r10 : 000000002FF3B010  r11 : 0000000000000000
r12 : 0800000180076A98  r13 : 0000000110003730  r14 : 0000000000000001
r15 : 00000000200FEB78  r16 : 00000000200FEB88  r17 : BADC0FFEE0DDF00D
r18 : BADC0FFEE0DDF00D  r19 : BADC0FFEE0DDF00D  r20 : BADC0FFEE0DDF00D
r21 : BADC0FFEE0DDF00D  r22 : BADC0FFEE0DDF00D  r23 : BADC0FFEE0DDF00D
r24 : BADC0FFEE0DDF00D  r25 : BADC0FFEE0DDF00D  r26 : BADC0FFEE0DDF00D
r27 : BADC0FFEE0DDF00D  r28 : BADC0FFEE0DDF00D  r29 : BADC0FFEE0DDF00D
r30 : BADC0FFEE0DDF00D  r31 : 0000000110000688
s0  : 60000000  s1  : 007FFFFF  s2  : 60010B68  s3  : 007FFFFF  s4  : 007FFFFF
s5  : 007FFFFF  s6  : 007FFFFF  s7  : 007FFFFF  s8  : 007FFFFF  s9  : 007FFFFF
s10 : 007FFFFF  s11 : 007FFFFF  s12 : 007FFFFF  s13 : 007FFFFF  s14 : 007FFFFF
s15 : 007FFFFF
prev      00000000 kjmpbuf   00000000 stackfix  2FF3B2A0 intpri    00
curid     00006FBC sralloc   A0000000 ioalloc   00000000 backt     00
flags     00 tid       00000000 excp_type 00000000
fpscr     00000000 fpeu            00 fpinfo          00 fpscrx    00000000
o_iar     00000000 o_toc     00000000 o_arg1    00000000
excbranch 00000000 o_vaddr   00000000 mstext    00062C08
Except : dar   08000001000581D4

KDB(0)>
```

## lastbackt subcommand

## Purpose

The **lastbackt** subcommand prints the context (Machine State Save Area) for when the last backtracking fault was taken on either the current processor or the specified processor.

## Syntax

**lastbackt** [*cpu*]

## Parameters

- *cpu* – Specifies a cpu index as a decimal value. If the cpu index is omitted, **lastbackt** defaults to the current cpu context.

## Aliases

No aliases.

## Example

The following is an example of how to use the **lastbackt** subcommand:

```
KDB(0)>lastbackt  //use current cpu context

Machine State Save Area
iar   : 0002599C  msr   : 00009030  cr    : 20000000  lr    : 000259B8
ctr   : 000258EC  xer   : 00000000  mq    : 00000000
r0  : 00000000  r1  : 2FF3B338  r2  : 002E65E0  r3  : 00000003  r4  : 00000002
r5  : 00000006  r6  : 002E6750  r7  : 00000000  r8  : DEADBEEF  r9  : DEADBEEF
r10 : DEADBEEF  r11 : 00000000  r12 : 00009030  r13 : DEADBEEF  r14 : DEADBEEF
r15 : DEADBEEF  r16 : DEADBEEF  r17 : DEADBEEF  r18 : DEADBEEF  r19 : DEADBEEF
r20 : DEADBEEF  r21 : DEADBEEF  r22 : DEADBEEF  r23 : DEADBEEF  r24 : DEADBEEF
r25 : DEADBEEF  r26 : DEADBEEF  r27 : DEADBEEF  r28 : 000034E0  r29 : 000C6158
r30 : 000C0578  r31 : 00005004
s0  : 00000000  s1  : 007FFFFF  s2  : 0000F00F  s3  : 007FFFFF  s4  : 007FFFFF
s5  : 007FFFFF  s6  : 007FFFFF  s7  : 007FFFFF  s8  : 007FFFFF  s9  : 007FFFFF
s10 : 007FFFFF  s11 : 007FFFFF  s12 : 007FFFFF  s13 : 0000C00C  s14 : 00004004
s15 : 007FFFFF
prev      00000000 kjmpbuf   00000000 stackfix  00000000 intpri    0B
curid     00000306 sralloc   E01E0000 ioalloc   00000000 backt     03
flags     00 tid       00000000 excp_type 00000000
fpscr     00000000 fpeu            00 fpinfo          00 fpscrx    00000000
o_iar     00000000 o_toc     00000000 o_arg1    00000000
excbranch 00000000 o_vaddr   00000000 mstext    00000000
Except :
 csr 2FEC6B78 dsisr 40000000  bit set: DSISR_PFT
 srval 000019DD dar 2FEC6B78 dsirr 00000106

KDB(0)> lastbackt 1  //use cpu 1

Machine State Save Area
iar   : 00038ED0  msr   : 00001030  cr    : 2A442424  lr    : 00038ED0
ctr   : 002BCC00  xer   : 00000000  mq    : 00000000
r0  : 60017017  r1  : 2FF3B300  r2  : 002E65E0  r3  : 00000000  r4  : 00000002
r5  : E60000BC  r6  : 00000109  r7  : 00000000  r8  : 000C0300  r9  : 00000001
r10 : 2FF3B380  r11 : 00000000  r12 : 00001030  r13 : 00000001  r14 : 2FF22F54
r15 : 2FF22F5C  r16 : DEADBEEF  r17 : DEADBEEF  r18 : 0000040F  r19 : 00000000
r20 : 00000000  r21 : 00000003  r22 : 01000001  r23 : 00000001  r24 : 00000000
r25 : E600014C  r26 : 000D1A08  r27 : 00000000  r28 : E3000160  r29 : E60000BC
r30 : 00000004  r31 : 00000004
s0  : 00000000  s1  : 007FFFFF  s2  : 0000A00A  s3  : 007FFFFF  s4  : 007FFFFF
s5  : 007FFFFF  s6  : 007FFFFF  s7  : 007FFFFF  s8  : 007FFFFF  s9  : 007FFFFF
s10 : 007FFFFF  s11 : 007FFFFF  s12 : 007FFFFF  s13 : 6001F01F  s14 : 00004004
```

```
s15 : 60004024
prev      00000000 kjmpbuf   00000000 stackfix  2FF3B300 intpri    00
curid     00000001 sralloc   E01E0000 ioalloc   00000000 backt     03
flags     00 tid       00000000 excp_type 00000000
fpscr     00000000 fpeu            00 fpinfo          00 fpscrx    00000000
o_iar     00000000 o_toc     00000000 o_arg1    00000000
excbranch 00000000 o_vaddr   00000000 mstext    00000000
Except :
 csr   30002F00 dsisr 40000000  bit set: DSISR_PFT
 srval 6000A00A dar   20022000 dsirr 00000106

KDB(0)>
```

## ttid subcommand

### Purpose

The **ttid** subcommand displays the thread table entry for a specific thread.

### Syntax

**ttid** [*tid*]

### Parameters

- *tid* – Specifies the thread ID. This value must be a decimal or a hexadecimal value as required by the *hexadecimal_wanted* toggle specified with the **set** subcommand. If no thread ID is specified, the entry for the current thread is displayed.

### Aliases

**th_tid**

### Example

The following is an example of how to use the **ttid** subcommand:

```
KDB(4)> p * //print process table
            SLOT NAME     STATE    PID  PPID  PGRP   UID  EUID  ADSPACE
...
proc+000100    1 init     ACTIVE 00001 00000 00000 00000 00000 0000A005
...
proc+000C00   12 gil      ACTIVE 00C18 00000 00000 00000 00000 00026013
...
KDB(4)> tpid 1 //print thread(s) of process pid 1
              SLOT NAME     STATE   TID PRI CPUID CPU FLAGS    WCHAN

thread+0000C0    1 init     SLEEP 001D9 03C       000 00000400
KDB(4)> ttid 001D9  //print thread with tid 0x1d9
              SLOT NAME     STATE   TID PRI CPUID CPU FLAGS    WCHAN

thread+0000C0    1 init     SLEEP 001D9 03C       000 00000400

NAME................ init
FLAGS............... WAKEONSIG
WTYPE............... WEVENT
............stackp64 :00000000  ..............stackp :2FF22DC0
...............state :00000003  ...............wtype :00000001
.............suspend :00000001  ...............flags :00000400
..............atomic :00000000
DATA................
...............procp :E3000100
...............userp :2FF3B6C0 <__ublock+0002C0>
............uthreadp :2FF3B400 <__ublock+000000>
THREAD LINK.........
..........prevthread :E60000C0
..........nextthread :E60000C0
SLEEP LOCK.........
............ulock64  :00000000  ..............ulock  :00000000
...............wchan :00000000  ..............wchan1 :00000000
...........wchan1sid :00000000  ........wchan1offset :01AB5A58
..............wchan2 :00000000  ..............swchan :00000000
...........eventlist :00000000  ..............result :00000000
.............polevel :000000AF  .............pevent  :00000000
.............wevent  :00000004  ..............slist  :00000000
...........lockcount :00000000
DISPATCH............
...............ticks :00000000  ...............prior :E60000C0
```

```
................next :E60000C0  ...............synch :FFFFFFFF
..............dispct :000008F6  ...............fpuct :00000000
SCHEDULER...........
...............cpuid :FFFFFFFF  ..............scpuid :FFFFFFFF
............affinity :00000001  .................pri :0000003C
..............policy :00000000  .................cpu :00000000
.............lockpri :0000003D  .............wakepri :0000007F
................time :000000FF  .............sav_pri :0000003C
SIGNAL..............
..............cursig :00000000
......(pending) sig  :
............sigmask  :
...............scp64 :00000000  .................scp :00000000
MISC................
............graphics :00000000  ..............cancel :00000000
...........lockowner :E60042C0  .............boosted :00000000
..............tsleep :FFFFFFFF
..........userdata64 :00000000  ............userdata :00000000
```

# tpid subcommand

## Purpose

The **tpid** subcommand displays all thread entries belonging to a process.

## Syntax

**tpid** [*pid*]

## Parameters

- *pid* – Specifies the process ID for which you want to display thread entries. This value must be a decimal or a hexadecimal value as required by the *hexadecimal_wanted* toggle specified with the set subcommand. If no process ID is specified, all thread table entries for the current process are displayed.

## Aliases

**th_pid**

## Example

The following is an example of how to use the **tpid** subcommand:

```
KDB(4)> p * //print process table
            SLOT NAME     STATE    PID  PPID  PGRP   UID  EUID  ADSPACE
...
proc+000100    1 init     ACTIVE 00001 00000 00000 00000 00000 0000A005
...
proc+000C00   12 gil      ACTIVE 00C18 00000 00000 00000 00000 00026013
...
KDB(4)> tpid 1 //print thread(s) of process pid 1
              SLOT NAME     STATE   TID PRI CPUID CPU FLAGS    WCHAN

thread+0000C0    1 init     SLEEP 001D9 03C       000 00000400
KDB(4)> tpid 00C18 //print thread(s) of process pid 0xc18
              SLOT NAME     STATE   TID PRI CPUID CPU FLAGS    WCHAN

thread+000900   12 gil      SLEEP 00C19 025       000 00001004
thread+000C00   16 gil      SLEEP 01021 025 00000 000 00003004 netisr_servers+000000
thread+000B40   15 gil      SLEEP 00F1F 025 00000 000 00003004 netisr_servers+000000
thread+000A80   14 gil      SLEEP 00E1D 025 00000 000 00003004 netisr_servers+000000
thread+0009C0   13 gil      SLEEP 00D1B 025 00000 000 00003004 netisr_servers+000000
```

## rq subcommand

### Purpose

The **rq** subcommand lists threads currently queued on the system run queues.

### Syntax

**rq** [ *bucket* | effectiveaddress]

### Parameters

- *bucket* – Lists all threads queued in a particular bucket across all run queues. The bucket is equal to the thread priority minus 1.
- *effectiveaddress* – Lists all threads queued in the bucket specified by the effective address.

If the **rq** subcommand is used with no parameters, a list of all buckets currently containing threads across all run queues is generated. If the **rq** subcommand is used with parameters, you can restrict the generated list to a particular run queue or to a particular bucket across all run queues.

### Aliases

**runq**

### Example

The following is an example of how to use the **rq** subcommand:

```
KDB(0)> rq
RQ                BUCKET HEAD            COUNT

02172D04     256 pvthread+000100     1
02172504     256 pvthread+000180     1
02173A1C      70 pvthread+005580     7
02173D04     256 pvthread+000200     1
02173504     256 pvthread+000280     1
KDB(0)> rq 02173A1C  //bucket address from the RQ column
LOCAL RUNQ(  2) ENTRY( 70) 02173A1C
                SLOT NAME     STATE    TID PRI  RQ CPUID  CL WCHAN

pvthread+005580  171>bash     RUN   00AB67 045   2         0
pvthread+004D00  154>bash     RUN   009A7F 045   2         0
pvthread+006100  194>bash     RUN   00C2B7 045   2         0
pvthread+006500  202>bash     RUN   00CAC9 045   2         0
pvthread+004C00  152>bash     RUN   009851 045   2         0
pvthread+006380  199>bash     RUN   00C701 045   2         0
pvthread+006280  197>bash     RUN   00C5B7 045   2         0
KDB(0)> rq 256  //bucket number from the RQ column
LOCAL RUNQ(  0) ENTRY(256) 02172D04
                SLOT NAME     STATE    TID PRI  RQ CPUID  CL WCHAN

pvthread+000100    2>wait     RUN   000205 0FF   0 00000   0
LOCAL RUNQ(  1) ENTRY(256) 02172504
                SLOT NAME     STATE    TID PRI  RQ CPUID  CL WCHAN

pvthread+000180    3>wait     RUN   000307 0FF   1 00001   0
LOCAL RUNQ(  2) ENTRY(256) 02173D04
                SLOT NAME     STATE    TID PRI  RQ CPUID  CL WCHAN

pvthread+000200    4>wait     RUN   000409 0FF   2 00002   0
LOCAL RUNQ(  3) ENTRY(256) 02173504
                SLOT NAME     STATE    TID PRI  RQ CPUID  CL WCHAN
```

```
pvthread+000280    5>wait     RUN   00050B 0FF   3 00003   0
GLOBAL RUNQ(node  0) ENTRY(256) 02171904
KDB(0)>
```

## rqi subcommand

### Purpose

The **rqi** subcommand displays information about run queues on the system.

### Syntax

**rqi** [ **-mrq** | *queue* | *slot* ]

### Parameters

- **-mrq** – Displays information about all mrq nodes in the system.
- *queue* – Specifies the effective address for the run queue structure specified by the effective address.
- *slot* – Specifies the run queue structure you want to display.

If the **rqi** subcommand is run without any parameters, a summary line for each run queue in the system is displayed. If the **rqi** subcommand is run with parameters, a specific run queue structure or the mrq nodes in the system are displayed.

### Aliases

**rqa**

### Example

The following is an example of how to use the **rqi** subcommand:

```
KDB(0)> rqi -mrq
primary_grq.................. 2171400
run_queue_max_local.......... 00000003  run_queue_max_global......... 00000080
num_nodes_onl................ 00000001  nodep @ 11EA710

MRQ_NODE @ 2171000
my_ndx........... 0000 rq_start_ndx..... 0000 lbolt........ 0006
active_rqs....... 0004 max_rqs.......... 0004
rqs_mask..... F0000000 00000000 00000000 00000000
S2_threshold..... 0000 num_S2........... 0001 S3_threshold. 00000180
thread_count..... 00A3 load......... 00000003 rq_slot.... @ 21711C8
sched_tid........ 00000003 reaper_tid....... 0000060D
zstart........... 0 zfinal........... E200D000
pref_S2id........  0
S2_stealable.....  0 0 0 0
S2id.............  0 0 0 0
num_S1........... 04
pref_S1id........ 00
S3_anysteals.....  0
S2_load.......... FFFFFFF4  balanced
S1_loads......... 00000000 00000000 00000000 00000000
KDB(0)>

KDB(0)> rqi
 RQ Node CPUs First Threads   stl  ustl any S1stl S2stl S3stl Busy Load

  0   0    1     0       38     0     0   0   0.0   1.1   0.0   0   0.0
  1   0    1     1       44     0     0   0   0.0   0.5   0.0   0   0.0
  2   0    1     2       42     0     0   0   0.0   1.0   0.0   0   0.0
  3   0    1     3       39     0     0   0   0.0   0.5   0.0   0   0.0
128   0    4     0        0     0     0   0                     0   0.0
KDB(0)> rqi 3  //slot number from RQ column in rqi subcommand
RUN_QUEUE @ 2173000
runrun............. 00000000 rq_stealable....... 00000000 S2_stealable... 00
rq_unstealable..... 00000000 rq_load............ 0000000F rq_S2id........ 00
rq_my_node_ndx......... 0000 rq_S1id................ 0003
```

```
rq_my_ndx............. 0003 rq_my_node_offset...... 0003
rq_cpu_start_ndx....... 0003 rq_cpu_node_offset..... 0003
rq_active_cpus......... 0001 rq_max_cpus........... 0001
rq_next_cpu........... 0000
rq_cpus_mask....... 80000000 00000000 00000000 00000000
rq_thread_count.... 00000027 rq_node_pointer.... 2171000
rq_busy_ticks.......... 0000 rq_busy............... 0000 rq_tload........ 0000
rq_best_run_pri/fixed.. FF/0 run_queue_lock..... 0
placement_load..... F
rq_steals_this_tick.... 0000 0000 0000 0000
rq_steals_this_second.. 0000 0000 0000 0000
rq_steal_smooth.... 00000000 00000000 00000083 00000000
dispct 007B7334 S0_misses 0000B6EB S1_misses 0000B6EB S2_misses 00000000
rq_lbolt............... 0052 rq_curthread_band...... 0000 stealing_active... 00
run_mask[0]........ 00000000 00000000 00000000 00000000
run_mask[4]........ 00000000 00000000 00000000 00000000
shared_S0.......... 00000000 00000000 00000000 00000000
shared_S1.......... 00000000 00000000 00000000 00000000
shared_S2.......... E0000000 00000000 00000000 00000000
thread_run....... @ 2173108
stealing_blocked... 00000000 00000000 00000000 00000000
banded_load[00]............ 00000000 00000000 00000000 00000000
banded_load[04]............ 00000000 00000000 00000000 00000000
banded_load[08]............ 00000000 00000000 00000000 00000000
banded_load[12]............ 00000000 00000000 00000000 00000000
banded_load_avg[00]........ 00000000 00000000 00000000 00000000
banded_load_avg[04]........ 00000000 00000000 00000000 00000000
banded_load_avg[08]........ 00000000 00000000 00000000 00000000
banded_load_avg[12]........ 00000000 00000000 00000000 00000000
KDB(0)>
```

## lq subcommand

### Purpose

The **lq** subcommand displays information about threads waiting on a lock.

### Syntax

**lq** [ *bucket* | *effectiveaddress* ]

### Parameters

- *bucket* – Displays information about a thread in the specified lock queue bucket.
- *effectiveaddress* – Displays information about a thread in the lock queue bucket that is specified by the effective address.

When run without any parameters, this subcommand displays a list of all threads which are currently waiting on some lock. With a parameter, the subcommand displays information about a waiting thread in a specific lock queue bucket.

### Aliases

**lockq**

### Example

The following is an example of how to use the **lq** subcommand:

```
KDB(0)> lq
                    BUCKET HEAD            COUNT

slist_table+0007E0    253 pvthread+003000     1
KDB(0)> lq 253 (lock queue bucket from the previous command)
SLIST_TABLE ENTRY(253): slist_table+0007E0
                SLOT NAME     STATE    TID PRI  RQ CPUID  CL WCHAN

pvthread+003000   96*v3fshelp SLEEP 006023 03E   2         0 inodes+3F48A64 slis
t_table+0007E0
KDB(0)>
```

# cr subcommand

## Purpose

The **cr** subcommand displays information about the checkpoint and the restart identifiers from the global *crid_table*.

## Syntax

**cr** [ ***** | **-i** *id* | *slot* | *effectiveaddress* ]

## Parameters

- ***** – Causes the **crid** subcommand to display a summary of all **crid** structures in the system.
- **-i** – Specifies the checkpoint or restart identifier (CRID) of the **crid** structure to be displayed.
- *slot* – Specifies the slot number within the *crid_table* of the **crid** structure to be displayed.
- *effectiveaddress* – Specifies the effective address of a particular **crid** structure to be displayed.

If the **cr** subcommand is run without any parameters, the **crid** structure is displayed for the current process if one exists. If the **cr** subcommand is run with parameters, a summary of all **crid** structures in the table are displayed or any specific **crid** structure is displayed.

## Aliases

**crid**

## Example

The following is an example of how to use the **cr** subcommand:

```
KDB(0)> cr 42
ADDRESS          SLOT ID       FLAGS    OWNER    CHKSYNCH

F10010F00406BA80   42 00000001 00000000 00000000 00000000

ID......... rcrid    :00000001 vcrid    :00000000
FLAGS...... flags    :00000000
OWNER...... owner    :00000000
VIRTUALS... lvpid    :0000000000000000
........... lvtid    :0000000000000000
........... lvseq    :00
CHECKPOINT. chksynch :0000000000000000
........... chkfile  :0000000000000000

MEMBERS.... procpv :0000000000000000
KDB(0)>
```

## svmon subcommand

### Purpose

The **svmon** subcommand displays information about the memory and paging space use on a per-process basis.

### Syntax

**svmon** [ **-p** *pid* | **-s** *slot* | **-a** *effectiveaddress* | ***** | **-** ]

### Parameters

- **-p** *pid* – Displays detailed information about the process specified by its process identifier.
- **-s** *slot* – Displays detailed information about the process in the specified process slot.
- **-a** *effectiveaddress* – Displays detailed information about the process specified by the effective address of its **pvproc** structure.
- ***** – Displays a brief summary about all the processes on the system when the asterisk ( * ) is the only parameter.
- **–** – Displays detailed information about all the processes on the system when the minus sign ( – ) is the only parameter.

When run without any parameters, the **svmon** subcommand displays information about the memory and paging space us of the running process on the current processor. With parameters, information about other processes or a brief summary of all processes can be displayed.

### Aliases

No aliases.

### Example

The following is an example of how to use the **svmon** subcommand:

```
(0)> svmon

-------------------------------------------------------------------------------
     Pid Command          64-bit Mthrd LPage Kproc  Uid
    8196 wait               Y     N     N     Y       0

    Vsid      Esid Type Description              LPage  Inuse   Pin Pgsp Virtual
       0         0 work kernel segment               -   6127  3762    0  6127
    7003  FFFFFFFF work application stack            -      1     1    0     1
    5002 F00000002 work process private              -     11     8    0    11

   Inuse      Pin     Pgsp  Virtual
    6139     3771        0     6139
(0)> svmon *
     Pid Command          Inuse      Pin     Pgsp  Virtual 64-bit Mthrd LPage
       0 swapper           6143     3771        0     6143      Y     N     N
       1 init              8200     3766        0     8187      N     N     N
    8196 wait              6139     3771        0     6139      Y     N     N
   12294 wait              6139     3771        0     6139      Y     N     N
   16392 wait              6139     3771        0     6139      Y     N     N
   20490 wait              6139     3771        0     6139      Y     N     N
   24588 reaper            6141     3770        0     6141      Y     N     N
   28686 lrud              6139     3770        0     6139      Y     N     N
   32784 xmdetd            6141     3770        0     6141      Y     N     N
   36882 vmptacrt          6141     3770        0     6141      Y     N     N
   40980 pilegc            6146     3771        0     6146      Y     Y     N
   45078 xmgc              6141     3770        0     6141      Y     N     N
   49176 netm              6141     3770        0     6141      Y     N     N
```

```
   53274 gil                6163      3774         0      6163       Y      Y      N
   57372 wlmsched           6141      3770         0      6141       Y      N      N
   65552 aixmibd            8188      3766         0      8116       N      N      N
   69706 cron               8082      3766         0      8061       N      N      N
   73900 random             6141      3770         0      6141       Y      N      N
   77870 jfsz               6141      3770         0      6141       Y      N      N
   81976 dog                6158      3774         0      6158       Y      Y      N
   86182 srcmstr            8093      3766         0      8080       N      N      N
   94322 errdemon           8256      3766         0      8154       N      N      N
   98366 lvmbb              6141      3770         0      6141       Y      N      N
  102462 kbiod              6146      3771         0      6146       Y      Y      N
  106598 syncd              8161      3779         0      8159       N      Y      N
  114922 snmpmibd64         6738      3769         0      6678       Y      N      N
  118862 portmap            8107      3766         0      8095       N      N      N
  127112 sendmail           8299      3766         0      8197       N      N      N
  131138 shlap64            6643      3769         0      6633       Y      N      N
  135240 rtcmd              6146      3771         0      6146       Y      Y      N
  139368 syslogd            8063      3766         0      8052       N      N      N
  143494 rmcd               8435      3768         0      8339       N      Y      N
  147664 hostmibd           8106      3766         0      8087       N      N      N
  151678 inetd              8069      3766         0      8059       N      N      N
  155778 muxatmd            8128      3766         0      8116       N      N      N
  159846 rpc.lockd          8048      3766         0      8046       N      N      N
  163994 rpc.statd          8206      3769         0      8185       N      Y      N
  168038 ksh                8161      3766         0      8106       N      N      N
  172130 biod               8002      3766         0      8001       N      N      N
  176260 IBM.AuditRMd       8359      3775         0      8306       N      Y      N
  184438 diagd              8069      3766         0      8060       N      N      N
  188594 qdaemon            8039      3766         0      8023       N      N      N
  192622 writesrv           8040      3766         0      8035       N      N      N
  196744 uprintfd           7997      3766         0      7995       N      N      N
  204906 rpc.lockd          6185      3779         0      6185       Y      Y      N
  213104 IBM.ServiceRM      8285      3774         0      8261       N      Y      N
  249980 IBM.ERrmd          8467      3775         0      8406       N      Y      N
  254120 kdb_64             7392      3769         0      6935       Y      N      N
  258180 IBM.CSMAgentR      8453      3777         0      8395       N      Y      N
(0)>
```

## meml subcommand

### Purpose

The **meml** subcommand displays information about the memory lock entries.

### Syntax

**meml** [[**-l**] | [**-e**] *effectiveaddress*]

### Parameters

- **-l** – Specifies the address of a memory lock entries list.
- **-e** – Specifies the address of a memory lock entry.
- *effectiveaddress* – Identifies the effective address. Use symbols, hexadecimal values, or hexadecimal expressions to specify the address.

### Aliases

**memlock**

### Example

The following is an example of how to use the **meml** subcommand:

```
KDB(0)> meml ?
MEML usage: meml [[-l|-e] eaddr][?]
          : meml -l to print a memlock list
          : meml -e to print a memlock list entry
KDB(0)> meml -l 3007A5C0

Memlock list, address 3007A5C0

Memlock list entry, address 3007A5C0
   next entry        (next)   : 000000003007AF60
   previous entry    (prev)   : 0000000000000000
   start address     (start)  : 0000000020000000
   number of bytes   (size)   : 0000000000011000


Memlock list entry, address 3007AF60
   next entry        (next)   : 0000000000000000
   previous entry    (prev)   : 000000003007A5C0
   start address     (start)  : 000000002DF22000
   number of bytes   (size)   : 0000000002001000
KDB(0)> meml -e 000000003007A5C0

Memlock list entry, address 3007A5C0
   next entry        (next)   : 000000003007AF60
   previous entry    (prev)   : 0000000000000000
   start address     (start)  : 0000000020000000
   number of bytes   (size)   : 0000000000011000
```

# cred subcommand

## Purpose

The **cred** subcommand displays the credentials structure for a specific effective address.

## Syntax

**cred** [*effectiveaddress*]

## Parameters

- *effectiveaddress* – Specifies the effective address of a credentials structure.

## Aliases

No aliases.

## Example

The following is an example of how to use the **cred** subcommand:

```
KDB(0)> cred F10006000BD42AFC
ref.........00000017 ruid........00000000 uid.........00000000
suid........00000000 luid........00000000 acctid......00000000
gid.........00000000 rgid........00000000 sgid........00000000
ngrps.......00000007 pag[0]......00000000
groups[00]..00000000 groups[01]..00000002 groups[02]..00000003
groups[03]..00000007 groups[04]..00000008 groups[05]..0000000A
groups[06]..0000000B
pag[01]..F100060000000000 pag[02]..0000000000000000
pag[03]..0000000000000000 pag[04]..0000000000000000
pag[05]..0000000000000000 pag[06]..0000000000000000
pag[07]..0000000000000000 pag[08]..0000000000000000
mpriv.......FFFFFFFF FFFFFFFF ipriv.......FFFFFFFF FFFFFFFF
epriv.......FFFFFFFF FFFFFFFF bpriv.......FFFFFFFF FFFFFFFF
ecap........00000000 00000000 icap........00000000 00000000
pcap........00000000 00000000
KDB(0)>
```

# Chapter 28. Display storage subsystem information subcommands

The subcommands in this category display storage subsystem information. These subcommands include the following:

# pbuf subcommand

## Purpose

The **pbuf** subcommand prints physical buffer information.

## Syntax

**pbuf** [*effectiveaddress*]

## Parameters

- *effectiveaddress* – Specifies the effective address of the physical buffer. Use symbols, hexadecimal values, or hexadecimal expressions to specify the address.

## Aliases

No aliases.

## Example

The following is an example of how to use the **pbuf** subcommand:

```
KDB(0)> pbuf 34D6A000
PBUF............ 34D6A000
pb............ @ 34D6A000 flags........... 000C8010
 SPLIT MPSAFE INITIAL
forw............ 000FB505 back............ 00000000
av_forw......... 35776400 av_back......... 00000000
iodone:  vm_pfend+000000
vp.............. 00000000 dev............. 000A0003
blkno........... 00008B70 bcount.......... 00001000
error........... 00000000 resid........... 00001000
work............ 00000000 options......... 00000000
event........... 00000000 start.tv_sec.... 403283C3
start.tv_nsec... 00000000
pb............ @ 34D6A000 pb_lbuf......... 00000000
pb_sched........ 00000000 pb_pvol......... 00000000
pb_bad.......... 00000000 pb_start........ 00000000
pb_mirror....... 00000000 pb_miravoid..... 00000000
pb_mirbad....... 00000000 pb_mirdone...... 00000000
pb_swretry...... 00000000 pb_type......... 00000000
pb_bbfixtype.... 00000000 pb_bbop......... 00000000
pb_bbstat....... 00000000 pb_whl_stop..... 00000000
pb_part......... 00000000 pb_bbcount...... 00000000
stripe_next..... 00000000 stripe_status... 00000000
orig_addr....... 00000000 orig_count...... 00000000
partial_stripe.. 00000000 first_issued.... 00000000
orig_bflags..... 00000000 pb_forw......... 0000 pb_back......... 0000
```

## volgrp subcommand

### Purpose

The **volgrp** subcommand displays volume group information. The **volgrp** structure addresses are registered in the **devsw** table in the **DSDPTR** field.

### Syntax

**volgrp** [*effectiveaddress*]

### Parameters

- *effectiveaddress* – Specifies the effective address of the **volgrp** structure to display. Use symbols, hexadecimal values or hexadecimal expressions to specify the address.

### Aliases

No aliases.

### Example

The following is an example of how to use the **volgrp** subcommand:

```
KDB(0)>  devsw 0a

Slot address 0571E280
MAJOR: 00A
   open:      01B44DE4
   close:     01B44470
   read:      01B43CD0
   write:     01B43C04
   ioctl:     01B42B18
   strategy:  .hd_strategy
   tty:       00000000
   select:    .nodev
   config:    01B413A0
   print:     .nodev
   dump:      .hd_dump
   mpx:       .nodev
   revoke:    .nodev
   dsdptr:    34D6C000
   selptr:    00000000
   opts:      0000000A       DEV_DEFINED DEV_MPSAFE

KDB(0)> volgrp 34D6C000
VOLGRP............. 34D6C000
vg_lock............ @ 34D6C000 vg_lock............... 00000000
partshift............ 00000010
open_count........... 00000009 flags................. 00000000
lvols.............. @ 34D6C02C
pvols.............. @ 34D6C82C major_num............. 0000000A
vg_id................ 0009FFFA00004C00000000F9E7859DCE
nextvg............... 00000000 opn_pin............. @ 34D6CA2C
von_pid.............. 00000C36 nxtactvg.............. 00000000
ca_freepvw........... 00000000 ca_pvwmem............. 00000000
ca_hld............. @ 34D6CA7C ca_pv_wrt........... @ 34D6CA88
ca_inflt_cnt......... 00000000 ca_size............... 00000000
ca_pvwblked.......... 00000000 mwc_rec............... 00000000
ca_part2............. 00000000 ca_lst................ 00000000
ca_hash............ @ 34D6CAAC bcachwait............. FFFFFFFF
ecachwait............ FFFFFFFF wait_cnt.............. 00000000
quorum_cnt........... 00000002 wheel_idx............. 00000000
whl_seq_num.......... 00000000 sa_act_lst............ 00000000
sa_hld_lst........... 00000000 vgsa_ptr.............. 34D6E000
```

```
config_wait........... FFFFFFFF sa_lbuf............. @ 34D6CB10
sa_pbuf............. @ 34D6CB68
sa_intlock.......... @ 34D6CC0C sa_intlock............ 00000000
vg_intlock.......... @ 34D6CC10 vg_intlock............ 00000000
refresh_Q........... @ 34D6CC14
gs_clvm............. @ 34D6CC20
oclvm............... @ 34D6CC24
ca_pvwaitq.......... @ 34D6CACC
LVOL[000]....... 3004AF00
work_Q.......... 00000000 lv_status....... 00000000
lv_options...... 00000001 nparts.......... 00000001
i_sched......... 00000000 nblocks......... 00200000
parts[0]........ 34D29A00 pvol@ 34D90C00 dev 00170001 start 00000000
parts[1]........ 00000000
parts[2]........ 00000000
maxsize......... 00000000 tot_rds......... 00000000
complcnt........ 00000000 waitlist........ FFFFFFFF
stripe_exp...... 00000000 striping_width.. 00000000
lvol_intlock. @  3004AF3C lvol_intlock.... 00000000
LVOL[001].......

...
```

## pvol subcommand

### Purpose

The **pvol** subcommand displays the physical volume data structure.

### Syntax

**pvol** [*effectiveaddress*]

### Parameters

- *effectiveaddress* – Specifies the effective address of the **pvol** structure to display. Use symbols, hexadecimal values, or hexadecimal expressions to specify the address.

### Aliases

No aliases.

### Example

The following is an example of how to use the **pvol** subcommand:

```
KDB(0)> pvol 34D6A000
PVOL............... 34D6A000
dev................ 000C8010 xfcnt.............. 00000000
pvstate............ 00000029
pvnum.............. FFFFD47C vg_num............. 00000000
fp................. 000A0003 flags.............. 00000000
num_bbdir_ent...... FFFF8B70 fst_usr_blk........ 0116D000
beg_relblk......... 00001000 next_relblk........ 000000001
max_relblk......... 00001000 defect_tbl......... 00000000
sa_area[0]....... @ 34D6A038
sa_area[1]....... @ 34D6A040 pv_pbuf.......... @ 34D6A048
oclvm............ @ 34D6A0F0
```

# lvol subcommand

## Purpose

The **lvol** subcommand displays logical volume information.

## Syntax

**lvol** [*effectiveaddress*]

## Parameters

- *effectiveaddress* – Specifies the effective address of the **lvol** structure to display. Use symbols, hexadecimal values, or hexadecimal expressions to specify the address.

## Aliases

No aliases.

## Example

The following is an example of how to use the **lvol** subcommand:

```
KDB(0)> lvol 3004AF00
LVOL............ 3004AF00
work_Q.......... 00000000 lv_status....... 00000000
lv_options...... 00000001 nparts.......... 00000001
i_sched......... 00000000 nblocks......... 00200000
parts[0]........ 34D29A00 pvol@ 34D90C00 dev 00170001 start 00000000
parts[1]........ 00000000
parts[2]........ 00000000
maxsize......... 00000000 tot_rds......... 00000000
complcnt........ 00000000 waitlist........ FFFFFFFF
stripe_exp...... 00000000 striping_width.. 00000000
lvol_intlock. @  3004AF3C lvol_intlock.... 00000000
```

## scd subcommand

### Purpose

The **scd** subcommand displays the **scdisk_diskinfo** structure.

### Syntax

**scd** [*slot* | *effectiveaddress*]

### Parameters

- *slot* – Specifies the slot number of the scdisk entry to be displayed. To use this parameter, the scdisk list must have been previously loaded using the **scd** subcommand with no parameter. This value must be a decimal number.
- *effectiveaddress* – Specifies the effective address of an **scdisk_diskinfo** structure to display. Use symbols, hexadecimal values, or hexadecimal expressions to specify the address.

If no argument is specified, the **scd** subcommand loads the slot numbers with addresses from the scdisk_list array. If the scdisk_list symbol cannot be located to load these values, the user is prompted for the address of the scdisk_list array. Obtain this address by locating the data address for the scdiskpin kernel extension and adding the offset to the scdisk_list array, which is obtained from a map, to that value.

A specific scdisk_list entry can be displayed by specifying either a slot number or the effective address of the entry. You can only use a slot number if the slots were previously loaded using the **scd** subcommand with no arguments.

### Aliases

**scdisk**

### Example

The following is an example of how to use the **scd** subcommand:

```
KDB(4)> lke 80  //print kernel extension information
     ADDRESS     FILE FILESIZE    FLAGS MODULE NAME

 80 05630900 01A57E60 0000979C 00000262  /etc/drivers/scdiskpin
le_flags........ TEXT DATAINTEXT DATA DATAEXISTS
le_fp........... 00000000
le_loadcount.... 00000000
le_usecount..... 00000001
le_data/le_tid.. 01A61320 <---    //this address plus the offset to
le_datasize..... 000002DC         //the scdisk_list array (from a map)
le_exports...... 0565E400         //are used to initialize the slots for
le_lex.......... 00000000         //the scd subcommand.
le_defered...... 00000000
le_filename..... 05630944
le_ndepend...... 00000001
le_maxdepend.... 00000001
le_de........... 00000000
KDB(4)> d 01A61320 100  //print data
01A61320: 0000 000B  0000 0006  FFFF FFFF  0562 7C00   .............b|.
01A61330: 0000 0000  0000 0000  0000 0000  0000 0000   ................
01A61340: 01A6 08DC  01A6 08D8  01A6 08D4  01A6 08D0   ................
01A61350: 01A6 08CC  01A6 08C8  01A6 08C4  01A6 08C0   ................
01A61360: 01A6 0920  01A6 0960  01A6 09A0  01A6 09E0   ... ...`........
01A61370: 01A6 0A20  01A6 0A60  01A6 0AA0  01A6 0AE0   ... ...`........
01A61380: 01A6 0B20  01A6 0B60  01A6 0BA0  01A6 0BE0   ... ...`........
01A61390: 01A6 0C20  01A6 0C60  01A6 0CA0  01A6 0CE0   ... ...`........
01A613A0: 7363 696E  666F 0000  6366 676C  6973 7400   scinfo..cfglist.
01A613B0: 6F70 6C69  7374 0000  4028 2329  3435 2020   oplist..@(#)45
```

```
01A613C0: 312E 3139  2E36 2E31  3620 2073  7263 2F62   1.19.6.16  src/b
01A613D0: 6F73 2F6B  6572 6E65  7874 2F64  6973 6B2F   os/kernext/disk/
01A613E0: 7363 6469  736B 622E  632C 2073  7973 7864   scdiskb.c, sysxd
01A613F0: 6973 6B2C  2062 6F73  3432 302C  2039 3631   isk, bos420, 961
01A61400: 3354 2031  2F38 2F39  3620 3233  3A34 313A   3T 1/8/96 23:41:
01A61410: 3538 0000  0000 0000  0567 4000  0567 5000   58.......g@..gP.
KDB(4)> scd  //print scsi disk table
Unable to find <scdisk_list>
Enter the scdisk_list address (in hex): 01A61418
Scsi pointer [01A61418]
slot  0...........05674000
slot  1...........05675000
slot  2...........0566C000
slot  3...........0566D000
slot  4...........0566E000
slot  5...........0566F000
slot  6...........05670000
slot  7...........05671000
slot  8...........05672000
slot  9...........05673000
slot 10...........0C40D000
slot 11...........00000000
slot 12...........00000000
slot 13...........00000000
slot 14...........00000000
slot 15...........00000000

KDB(4)> scd 0  //print scsi disk slot 0
Scdisk info [05674000]
next......................00000000 next_open.................00000000
devno.....................00120000 adapter_devno.............00100000
watchdog_timer.watch.@....05674010 watchdog_timer.pointer....05674000
scsi_id...................00000000 lun_id....................00000000
reset_count...............00000000 dk_cmd_q_head.............00000000
dk_cmd_q_tail.............00000000 ioctl_cmd@................05674034
cmd_pool..................05628400 pool_index................00000000
open_event................FFFFFFFF checked_cmd...............00000000
writev_err_cmd............00000000 reassign_err_cmd..........00000000
reset_cmd@................056740FC reqsns_cmd@...............056741AC
writev_cmd@...............0567425C q_recov_cmd@..............0567430C
reassign_cmd@.............056743BC dmp_cmd@..................0567446C
dk_bp_queue@..............0567451C mode......................00000001
disk_intrpt...............00000000 raw_io_intrpt.............00000000
ioctl_chg_mode_flg........00000000 m_sense_status............00000000
opened....................00000001 cmd_pending...............00000000
errno.....................00000000 retain_reservation........00000000
q_type....................00000000 q_err_value...............00000001
clr_q_on_error............00000001 buffer_ratio..............00000000
cmd_tag_q.................00000000 q_status..................00000000
q_clr.....................00000000 timer_status..............00000000
restart_unit..............00000000 retry_flag................00000000
(4)> more (^C to quit) ?  //continue
safe_relocate.............00000000 async_flag................00000000
dump_inited...............00000001 extended_rw...............00000001
reset_delay...............00000002 starting_close............00000000
reset_failures............00000000 wprotected................00000000
reserve_lock..............00000001 prevent_eject.............00000000
cfg_prevent_ej............00000000 cfg_reserve_lck...........00000001
load_eject_alt............00000000 pm_susp_bdr...............00000000
dev_type..................00000001 ioctl_pending.............00000000
play_audio................00000000 overide_pg_e..............00000000
cd_mode1_code.............00000000 cd_mode2_form1_code.......00000000
cd_mode2_form2_code.......00000000 cd_da_code................00000000
current_cd_code...........00000000 current_cd_mode...........00000001
multi_session.............00000000 valid_cd_modes............00000000
mult_of_blksize...........00000001 play_audio_started........00000000
rw_timeout................0000001E fmt_timeout...............00000000
```

```
start_timeout.............0000003C reassign_timeout..........00000078
queue_depth...............00000001 cmds_out..................00000000
raw_io_cmd................00000000 currbuf...................0A0546E0
low.......................0A14E3C0 block_size................00000200
cfg_block_size............00000200 last_ses_pvd_lba..........00000000
max_request...............00040000 max_coalesce..............00010000
lock......................FFFFFFFF fp........................00414348
(4)> more (^C to quit) ?  //continue
error_rec@................05674598 stats@....................05674648
mode_data_length..........0000003D disc_info@................0567465C
mode_buf@.................05674660 sense_buf@................05674760
ch_data@..................05674860 df_data@..................05674960
def_list_header@..........05674A60 ioctl_buf@................05674A64
mode_page_e@..............05674B63 dd@.......................05674B6C
df@.......................05674BB4 ch@.......................05674BFC
cd@.......................05674C44 ioctl_req_sense@..........05674C8C
capacity@.................05674CA4 def_list@.................05674CAC
dkstat@...................05674CB4
spin_lock@................05674CF8 spin_lock.................E80039A0
pmh@......................05674CFC pm_pending................00000000
pm_reserve@...............05674D41 pm_device_id..............00100000
pm_event..................FFFFFFFF pm_timer@.................05674D4C
KDB(4)> file 00414348  //print file (fp)
                 COUNT           OFFSET     DATA TYPE   FLAGS

  18 file+000330     1 0000000000000000 0BC4A950 GNODE  WRITE

f_flag......... 00000002 f_count........ 00000001
f_msgcount......... 0000 f_type............. 0003
f_data......... 0BC4A950 f_offset... 0000000000000000
f_dir_off...... 00000000 f_cred......... 00000000
f_lock@........ 00414368 f_lock......... E88007C0
f_offset_lock@. 0041436C f_offset_lock.. E88007E0
f_vinfo........ 00000000 f_ops.......... 001F3CD0 gno_fops+000000
GNODE.......... 0BC4A950
gn_seg....... 007FFFFF gn_mwrcnt.... 00000000 gn_mrdcnt.... 00000000
gn_rdcnt..... 00000000 gn_wrcnt..... 00000002 gn_excnt..... 00000000
gn_rshcnt.... 00000000 gn_ops....... 00000000 gn_vnode..... 00000000
gn_reclk..... 00000000 gn_rdev...... 00100000
gn_chan...... 00000000 gn_filocks... 00000000 gn_data...... 0BC4A940
gn_type...... BLK      gn_flags.....
KDB(4)> buf 0A0546E0  //print current buffer (currbuf)
                  DEV    VNODE    BLKNO FLAGS

  0 0A0546E0 00120000 00000000 00070A58 READ SPLIT MPSAFE MPSAFE_INITIAL

forw     00000000 back     00000000 av_forw  0A05DC60 av_back  0A14E3C0
blkno    00070A58 addr     00626000 bcount   00001000 resid    00000000
error    00000000 work     00000000 options  00000000 event    FFFFFFFF
iodone:  019057D4
start.tv_sec       00000000 start.tv_nsec      00000000
xmemd.aspace_id    00000000 xmemd.xm_flag      00000000 xmemd.xm_version   00000000
xmemd.subspace_id  00800802 xmemd.subspace_id2 00000000 xmemd.uaddr        00000000
```

# Chapter 29. Display memory allocation information subcommands

The subcommands in this category display memory allocation information. These subcommands include the following:

- heap
- xm
- kmbucket
- kmstats

# heap subcommand

## Purpose

The **heap** subcommand displays information about heaps.

## Syntax

**heap** *effectiveaddress*

## Parameters

- *effectiveaddress* – Specifies the effective address of the heap. Use symbols, hexadecimal values, or hexadecimal expressions to specify the address.

If no parameter is used, information is displayed for the kernel heap.

## Aliases

**hp**

## Example

The following is an example of how to use the **hp** alias for the **heap** subcommand:

```
KDB(2)> hp //print kernel heap information
Pinned heap 0FFC4000
sanity..... 48454150 base....... F11B7000
lock@...... 0FFC4008 lock....... 00000000
alt........ 00000001 numpages... 0000EE49
amount..... 002D2750 pinflag.... 00000001
newheap.... 00000000 protect.... 00000000
limit...... 00000000 heap64..... 00000000
vmrelflag.. 00000000 rhash...... 00000000
pagtot..... 00000000 pagused.... 00000000
frtot[00].. 00000000 [01].. 00000000 [02].. 00000000 [03].. 00000000
frtot[04].. 00000000 [05].. 00000000 [06].. 00000000 [07].. 00000000
frtot[08].. 00000000 [09].. 00000000 [10].. 00000000 [11].. 00000000
frused[00]. 00000000 [01].. 00000000 [02].. 00000000 [03].. 00000000
frused[04]. 00000000 [05].. 00000000 [06].. 00000000 [07].. 00000000
frused[08]. 00000000 [09].. 00000000 [10].. 00000000 [11].. 00000000
fr[00]..... 00FFFFFF [01].. 00FFFFFF [02].. 00FFFFFF [03].. 00FFFFFF
fr[04]..... 00003C22 [05].. 00004167 [06].. 00004A05 [07].. 00004845
fr[08]..... 000043B5 [09].. 00000002 [10].. 0000443A [11].. 00004842
Kernel heap 0FFC40B8
sanity..... 48454150 base....... F11B6F48
lock@...... 0FFC40C0 lock....... 00000000
alt........ 00000000 numpages... 0000EE49
amount..... 04732CF0 pinflag.... 00000000
newheap.... 00000000 protect.... 00000000
limit...... 00000000 heap64..... 00000000
vmrelflag.. 00000000 rhash...... 00000000
pagtot..... 00000000 pagused.... 00000000
frtot[00].. 00000000 [01].. 00000000 [02].. 00000000 [03].. 00000000
frtot[04].. 00000000 [05].. 00000000 [06].. 00000000 [07].. 00000000
frtot[08].. 00000000 [09].. 00000000 [10].. 00000000 [11].. 00000000
frused[00]. 00000000 [01].. 00000000 [02].. 00000000 [03].. 00000000
frused[04]. 00000000 [05].. 00000000 [06].. 00000000 [07].. 00000000
frused[08]. 00000000 [09].. 00000000 [10].. 00000000 [11].. 00000000
fr[00]..... 00FFFFFF [01].. 00FFFFFF [02].. 00FFFFFF [03].. 00FFFFFF
fr[04]..... 000049E9 [05].. 00003C26 [06].. 0000484E [07].. 00004737
fr[08]..... 00003C0A [09].. 00004A07 [10].. 00004855 [11].. 00004A11
addr...... 0000000000000000 maxpages.......... 00000000
peakpage.......... 00000000 limit_callout..... 00000000
newseg_callout.... 00000000 pagesoffset....... 0FFC4194
```

```
pages_sid......... 00000000
Heap anchor
... 0FFC4190 pageno FFFFFFFF pages.type.. 00 allocpage     offset... 00004A08
Heap Free list
... 0FFD69B4 pageno 00004A08 pages.type.. 02 freepage      offset... 00004A0C
... 0FFD69C4 pageno 00004A0C pages.type.. 03 freerange     offset... 00004A17
... 0FFD69C8 pageno 00004A0D pages.type.. 04 freesize      size..... 00000005
... 0FFD69D4 pageno 00004A10 pages.type.. 05 freerangeend  offset... 00004A0C
... 0FFD69F0 pageno 00004A17 pages.type.. 03 freerange     offset... NO_PAGE
... 0FFD69F4 pageno 00004A18 pages.type.. 04 freesize      size..... 0000A432
... 0FFFFAB4 pageno 0000EE48 pages.type.. 05 freerangeend  offset... 00004A17
Heap Alloc list
... 0FFC41B0 pageno 00000007 pages.type.. 01 allocrange    offset... NO_PAGE
... 0FFC41B4 pageno 00000008 pages.type.. 06 allocsize     size..... 00001E00
... 0FFCB9AC pageno 00001E06 pages.type.. 07 allocrangeend offset... 00000007
... 0FFCB9B0 pageno 00001E07 pages.type.. 01 allocrange    offset... NO_PAGE
... 0FFCB9B4 pageno 00001E08 pages.type.. 06 allocsize     size..... 00001E00
... 0FFD31AC pageno 00003C06 pages.type.. 07 allocrangeend offset... 00001E07
... 0FFD31B4 pageno 00003C08 pages.type.. 01 allocrange    offset... 00003C42
... 0FFD31B8 pageno 00003C09 pages.type.. 06 allocsize     size..... 00000002
... 0FFD31C4 pageno 00003C0C pages.type.. 01 allocrange    offset... NO_PAGE
... 0FFD31C8 pageno 00003C0D pages.type.. 06 allocsize     size..... 00000009
... 0FFD31E4 pageno 00003C14 pages.type.. 07 allocrangeend offset... 00003C0C
... 0FFD31E8 pageno 00003C15 pages.type.. 01 allocrange    offset... NO_PAGE
... 0FFD31EC pageno 00003C16 pages.type.. 06 allocsize     size..... 00000009
... 0FFD3208 pageno 00003C1D pages.type.. 07 allocrangeend offset... 00003C15
... 0FFD320C pageno 00003C1E pages.type.. 01 allocrange    offset... NO_PAGE
...
KDB(3)> dw msg_heap 8 //look at message heap
msg_heap+000000: 0000A02A CFFBF0B8 0000B02B CFFBF0B8  ...*.......+....
msg_heap+000010: 0000C02C CFFBF0B8 0000D02D CFFBF0B8  ...,.......-....
KDB(3)> mr s12           //set SR12 with message heap SID
s12 : 007FFFFF = 0000A02A
KDB(3)> heap CFFBF0B8          //print message heap
Heap CFFBF000
sanity..... 48454150 base....... F0041000
lock@...... CFFBF008 lock....... 00000000
alt........ 00000001 numpages... 0000FFBF
amount..... 00000000 pinflag.... 00000000
newheap.... 00000000 protect.... 00000000
limit...... 00000000 heap64..... 00000000
vmrelflag.. 00000000 rhash...... 00000000
pagtot..... 00000000 pagused.... 00000000
frtot[00].. 00000000 [01].. 00000000 [02].. 00000000 [03].. 00000000
frtot[04].. 00000000 [05].. 00000000 [06].. 00000000 [07].. 00000000
frtot[08].. 00000000 [09].. 00000000 [10].. 00000000 [11].. 00000000
frused[00]. 00000000 [01].. 00000000 [02].. 00000000 [03].. 00000000
frused[04]. 00000000 [05].. 00000000 [06].. 00000000 [07].. 00000000
frused[08]. 00000000 [09].. 00000000 [10].. 00000000 [11].. 00000000
fr[00]..... 00FFFFFF [01].. 00FFFFFF [02].. 00FFFFFF [03].. 00FFFFFF
fr[04]..... 00FFFFFF [05].. 00FFFFFF [06].. 00FFFFFF [07].. 00FFFFFF
fr[08]..... 00FFFFFF [09].. 00FFFFFF [10].. 00FFFFFF [11].. 00FFFFFF
Heap CFFBF0B8
sanity..... 48454150 base....... F0040F48
lock@...... CFFBF0C0 lock....... 00000000
alt........ 00000000 numpages... 0000FFBF
amount..... 00000100 pinflag.... 00000000
newheap.... 00000000 protect.... 00000000
limit...... 00000000 heap64..... 00000000
vmrelflag.. 00000000 rhash...... 00000000
pagtot..... 00000000 pagused.... 00000000
frtot[00].. 00000000 [01].. 00000000 [02].. 00000000 [03].. 00000000
frtot[04].. 00000000 [05].. 00000000 [06].. 00000000 [07].. 00000000
frtot[08].. 00000000 [09].. 00000000 [10].. 00000000 [11].. 00000000
frused[00]. 00000000 [01].. 00000000 [02].. 00000000 [03].. 00000000
frused[04]. 00000000 [05].. 00000000 [06].. 00000000 [07].. 00000000
frused[08]. 00000000 [09].. 00000000 [10].. 00000000 [11].. 00000000
```

```
fr[00]..... 00FFFFFF [01].. 00FFFFFF [02].. 00FFFFFF [03].. 00FFFFFF
fr[04]..... 00FFFFFF [05].. 00FFFFFF [06].. 00FFFFFF [07].. 00FFFFFF
fr[08]..... 00000000 [09].. 00FFFFFF [10].. 00FFFFFF [11].. 00FFFFFF
addr...... 0000000000000000 maxpages.......... 00000000
peakpage.......... 00000000 limit_callout..... 00000000
newseg_callout.... 00000000 pagesoffset....... 00000194
pages_sid......... 00000000
Heap anchor
... CFFBF190 pageno FFFFFFFF pages.type.. 00 allocpage     offset... 00000001
Heap Free list
... CFFBF198 pageno 00000001 pages.type.. 03 freerange     offset... NO_PAGE
... CFFBF19C pageno 00000002 pages.type.. 04 freesize      size..... 0000FFBE
... CFFFF08C pageno 0000FFBE pages.type.. 05 freerangeend  offset... 00000001
Heap Alloc list
KDB(3)> mr s12 //reset SR12
s12 : 0000A02A = 007FFFFF
```

## xmalloc subcommand

### Purpose

The **xmalloc** subcommand displays memory allocation information, finds the memory location of any heap record using the page index or finds the heap record using the allocated memory location.

### Syntax

**xmalloc** [**-s** [*effectiveaddress*]] [**-h** [*effectiveaddress*]] [[**-l**] **-f**] [[**-l**] **-a**] [[**-l**] **-p** *page*] [**-d** [*effectiveaddress*]] [**-v**] [[**-q**] **-u** [*size*]] [**-S**] [*effectiveaddress*] [**-H** *heap_addr*]

### Parameters

- **-s** – Displays allocation records matching the value of the *effectiveaddress* parameter. If the *effectiveaddress* parameter is not specified, the value of the **Debug_addr** symbol is used.
- **-h** – Displays free list records matching *effectiveaddress*. If *effectiveaddress* is not specified, the value of the **Debug_addr** symbol is used.
- **-l** – Enables verbose output. Applicable only with the **-f**, **-a**, and **-p** flags.
- **-f** – Displays records on the free list, from the first freed record to the last freed record.
- **-a** – Displays allocation records.
- **-p** *page* – Displays page information for the specified page. The page number is a hexadecimal value.
- **-d** – Displays the allocation record hash chain associated with the record hash value for the *effectiveaddress* parameter. If the *effectiveaddress* parameter is not specified, the value of the **Debug_addr** symbol is used.
- **-v** – Verifies allocation trailers for allocated records and verifies free fill patterns for free records.
- **-q** – Indicates that allocations should not be separated into size groups.
- **-u** – Displays heap statistics.
- *size* – Specifies the largest size allocation reported.
- **-S** – Displays heap locks and per-processor lists.

  **Note:** The per-processor lists are only used for the kernel heaps.
- *effectiveaddress* – Specifies the effective address for which information is to be displayed. Use symbols, hexadecimal values, or hexadecimal expressions to specify the effective address.
- **-H** *heap_addr* – Specifies the effective address of the heap for which information is displayed. If the **-H** parameter is not specified, information is displayed for the kernel heap. The **-H** parameter can be supplied with other **xmalloc** parameters. Use symbols, hexadecimal values, or hexadecimal expressions to specify the effective address.

Other than the **-u** parameter, these parameters require that the Memory Overlay Detection System (MODS) is active. If parameters require a memory address and no value is specified, the value of the **Debug_addr** symbol is used. If a system crash is caused by detection of a problem within MODS, this value is updated by MODS. The default heap reported on is the kernel heap.

### Aliases

**xm**

### Example

The following is an example of how to use the **xm** alias of the **xmalloc** subcommand:

```
(0)> stat
RS6K_SMP_MCA POWER_PC POWER_604 machine with 8 processor(s)
.......... SYSTEM STATUS
sysname... AIX        nodename.. jumbo32
```

```
release... 3          version... 4
machine... 00920312A0 nid....... 920312A0
time of crash: Fri Jul 11 08:07:01 1997
age of system: 1 day, 20 hr., 31 min., 17 sec.
.......... PANIC STRING
Memdbg: *w == pat

(0)> xm -s //Display debug xmalloc status
Debug kernel error message: The xmfree service has found data written beyond the
 end of the memory buffer that is being freed.
Address at fault was 0x09410200

(0)> xm -h 0x09410200  //Display debug xmalloc records associated with addr
0B78DAB0: addr......... 09410200 req_size..... 128 freed unpinned
0B78DAB0: pid.......... 00043158 comm......... bcross
Trace during xmalloc()                  Trace during xmfree()
002329E4(.xmalloc+0000A8)               002328F0(.xmfree+0000FC)
00235CD4(.dlistadd+000040)              00234F04(.setbitmaps+0001BC)
00235520(.newblk+00006C)                00236894(.finicom+0001A4)

0B645120: addr......... 09410200 req_size..... 128 freed unpinned
0B645120: pid.......... 0007DCAC comm......... bcross
Trace during xmalloc()                  Trace during xmfree()
002329E4(.xmalloc+0000A8)               002328F0(.xmfree+0000FC)
00235CD4(.dlistadd+000040)              00236614(.logdfree+0001E8)
00236574(.logdfree+000148)              00236720(.finicom+000030)

0B7A3750: addr......... 09410200 req_size..... 128 freed unpinned
0B7A3750: pid.......... 000010BA comm......... syncd
Trace during xmalloc()                  Trace during xmfree()
002329E4(.xmalloc+0000A8)               002328F0(.xmfree+0000FC)
00235CD4(.dlistadd+000040)              00234F04(.setbitmaps+0001BC)
00235520(.newblk+00006C)                00236894(.finicom+0001A4)

0B52B330: addr......... 09410200 req_size..... 128 freed unpinned
0B52B330: pid.......... 00058702 comm......... bcross
Trace during xmalloc()                  Trace during xmfree()
002329E4(.xmalloc+0000A8)               002328F0(.xmfree+0000FC)
00235CD4(.dlistadd+000040)              00236698(.logdfree+00026C)
00236510(.logdfree+0000E4)              00236720(.finicom+000030)

07A33840: addr......... 09410200 req_size..... 133 freed unpinned
07A33840: pid.......... 00042C24 comm......... ksh
Trace during xmalloc()                  Trace during xmfree()
002329E4(.xmalloc+0000A8)               002328F0(.xmfree+0000FC)
00271F28(.ld_pathopen+000160)           00271D24(.ld_pathclear+00008C)
0027FB6C(.ld_getlib+000074)             002ABF04(.ld_execload+00075C)

0B796480: addr......... 09410200 req_size..... 133 freed unpinned
0B796480: pid.......... 0005C2E0 comm......... ksh
Trace during xmalloc()                  Trace during xmfree()
002329E4(.xmalloc+0000A8)               002328F0(.xmfree+0000FC)
00271F28(.ld_pathopen+000160)           00271D24(.ld_pathclear+00008C)
0027FB6C(.ld_getlib+000074)             002ABF04(.ld_execload+00075C)

07A31420: addr......... 09410200 req_size..... 135 freed unpinned
07A31420: pid.......... 0007161A comm......... ksh
Trace during xmalloc()                  Trace during xmfree()
002329E4(.xmalloc+0000A8)               002328F0(.xmfree+0000FC)
00271F28(.ld_pathopen+000160)           00271D24(.ld_pathclear+00008C)
0027FB6C(.ld_getlib+000074)             002ABF04(.ld_execload+00075C)

07A38630: addr......... 09410200 req_size..... 125 freed unpinned
07A38630: pid.......... 0001121E comm......... ksh
Trace during xmalloc()                  Trace during xmfree()
002329E4(.xmalloc+0000A8)               002328F0(.xmfree+0000FC)
00271F28(.ld_pathopen+000160)           00271D24(.ld_pathclear+00008C)
```

```
0027FB6C(.ld_getlib+000074)            002ABF04(.ld_execload+00075C)

07A3D240: addr......... 09410200 req_size..... 133 freed unpinned
07A3D240: pid.......... 0000654C comm......... ksh
Trace during xmalloc()                 Trace during xmfree()
002329E4(.xmalloc+0000A8)              002328F0(.xmfree+0000FC)
00271F28(.ld_pathopen+000160)          00271D24(.ld_pathclear+00008C)
0027FB6C(.ld_getlib+000074)            002ABF04(.ld_execload+00075C)

(0)> heap
...
Heap Alloc list
... 0FFC41B0 pageno 00000007 pages.type.. 01 allocrange    offset... NO_PAGE
... 0FFC41B4 pageno 00000008 pages.type.. 06 allocsize     size..... 00001E00
... 0FFCB9AC pageno 00001E06 pages.type.. 07 allocrangeend offset... 00000007
... 0FFCB9B0 pageno 00001E07 pages.type.. 01 allocrange    offset... NO_PAGE
... 0FFCB9B4 pageno 00001E08 pages.type.. 06 allocsize     size..... 00001E00
... 0FFD31AC pageno 00003C06 pages.type.. 07 allocrangeend offset... 00001E07
... 0FFD31B4 pageno 00003C08 pages.type.. 01 allocrange    offset... 00003C42
... 0FFD31B8 pageno 00003C09 pages.type.. 06 allocsize     size..... 00000002
... 0FFD31C4 pageno 00003C0C pages.type.. 01 allocrange    offset... NO_PAGE
... 0FFD31C8 pageno 00003C0D pages.type.. 06 allocsize     size..... 00000009
... 0FFD31E4 pageno 00003C14 pages.type.. 07 allocrangeend offset... 00003C0C
...
(0)> xm -l -p 00001E07 //how to find memory address of heap index 00001E07
type..................... 1 (P_allocrange)
page_addr................ 02F82000 pinned................... 0
size..................... 00000000 offset................... 00FFFFFF
page_descriptor_address.. 0FFCB9B0
(0)> xm -l 02F82000 //how to find page index in kernel heap of 02F82000
P_allocrange (range of 2 or more allocated full pages)
page........... 00001E07 start.......... 02F82000 page_cnt....... 00001E00
allocated_size. 01E00000 pinned......... unknown
(0)> xm -l -p 00003C08 //how to find memory address of heap index 00003C08
type..................... 1 (P_allocrange)
page_addr................ 04D83000 pinned................... 0
size..................... 00000000 offset................... 00003C42
page_descriptor_address.. 0FFD31B4
(0)> xm -l 04D83000 //how to find page index in kernel heap of 04D83000
P_allocrange (range of 2 or more allocated full pages)
page........... 00003C08 start.......... 04D83000 page_cnt....... 00000002
allocated_size. 00002000 pinned......... unknown
```

## kmbucket subcommand

### Purpose

The **kmbucket** subcommand prints kernel memory allocator buckets.

### Syntax

**kmbucket** [**-l**] [**-c** *cpu*] [**-i** *index*] [*effectiveaddress*]

**kmbucket -k** *effectiveaddress*

**kmbucket -s**

### Parameters

- **-l** – Displays the bucket free list.
- **-c** *cpu* – Displays only buckets for the specified processor. Specify the cpu parameter as a decimal value.
- **-i** *index* – Displays only the bucket for the specified index. The index is specified as a decimal value.
- *effectiveaddress* – Specifies the effective address of the kmembucket structure to display. Use symbols, hexadecimal values, or hexadecimal expressions to specify the address.
- **-k** – Displays the **kmemusage** structure associated with the *effectiveaddress*.
- **-s** – Displays the **netkmem** structure.

If no arguments are specified, information is displayed for all allocator buckets for each processor.

### Aliases

**bucket**

### Example

The following is an example of how to use the **kmbucket** subcommand:

```
KDB(0)> kmbucket -c 0 -i 11

displaying kmembucket for cpu 0 offset 11 size 0x00000800

address...............F10006000BD8BD48  b_next..(x)...........F100061002AD1000
b_calls..(x)..........0000000000001405  b_total..(x)..........000000000000080A
b_totalfree..(x)......0000000000000006  b_elmpercl..(x).......0000000000000002
b_highwat..(x)........00000000000007AD  b_couldfree (sic).(x).0000000000000000
b_failed..(x).........0000000000000000  b_delayed.............0000000000000000
lock............... @ F10006000BD8BD90  lock..(x).............0000000000000000
delta.................FFFFFFFFFFFFD800
KDB(0)> kmbucket F10006000BD8BD48 //address field from above

displaying kmembucket for cpu 0 offset 11 size 0x00000800

address...............F10006000BD8BD48  b_next..(x)...........F100061002ACB000
b_calls..(x)..........0000000000001407  b_total..(x)..........000000000000080A
b_totalfree..(x)......0000000000000005  b_elmpercl..(x).......0000000000000002
b_highwat..(x)........00000000000007AD  b_couldfree (sic).(x).0000000000000000
b_failed..(x).........0000000000000000  b_delayed.............0000000000000000
lock............... @ F10006000BD8BD90  lock..(x).............0000000000000000
delta.................FFFFFFFFFFFFE000

Bucket free list.....
  1 next........F100061002ACB000 prev...00000000,
    kmemusage...F10006000BE08308 [000B 0002 00000000]
  2 next........F100061002AE0800 prev...F100061002ACB000,
```

```
    kmemusage...F10006000BE08500 [000B 0001 00000000]
  3 next........F100061002AC8000 prev...F100061002AE0800,
    kmemusage...F10006000BE082C0 [000B 0002 00000000]
  4 next........F100061002AC8800 prev...F100061002AC8000,
    kmemusage...F10006000BE082C0 [000B 0002 00000000]
  5 next........F100061002ACB800 prev...F100061002AC8800,
    kmemusage...F10006000BE08308 [000B 0002 00000000]
KDB(0)> kmbucket -k F100061002ACB000  //one of the next fields from above
This address belongs to the following kmemusage structure :
  kmemusage address.....F10006000BE08308
  ku_indx.....0000000B  free/page cnt.....00000002  ku_cpu.....00000000
KDB(0)>
```

## kmstats subcommand

## Purpose

The **kmstats** subcommand prints kernel allocator memory statistics.

## Syntax

**kmstats** [*effectiveaddress*]

## Parameters

- *effectiveaddress* – Specifies the effective address of the kernel allocator memory statistics entry to display. Use symbols, hexadecimal values, or hexadecimal expressions to specify the address.

If no address is specified, all of the kernel allocator memory statistics are displayed. If an address is entered, only the specified statistics entry is displayed.

## Aliases

No aliases.

## Example

The following is an example of how to use the **kmstats** subcommand:

```
KDB(0)> kmstats //print allocator statistics

displaying kmemstats for offset 0 free
address.................0025C120
inuse..(x)..............00000000
calls..(x)..............00000000
memuse..(x).............00000000
limit blocks..(x)........00000000
map blocks..(x).........00000000
maxused..(x)............00000000
limit..(x)..............02666680
failed..(x).............00000000
lock..(x)...............00000000

displaying kmemstats for offset 1 mbuf
address.................0025C144
inuse..(x)..............0000000D
calls..(x)..............002C4E54
memuse..(x).............00000D00
limit blocks..(x)........00000000
map blocks..(x).........00000000
maxused..(x)............0001D700
limit..(x)..............02666680
(0)> more (^C to quit) ? //continue
failed..(x).............00000000
lock..(x)...............00000000

displaying kmemstats for offset 2 mcluster
address.................0025C168
inuse..(x)..............00000002
calls..(x)..............00023D4E
memuse..(x).............00000900
limit blocks..(x)........00000000
map blocks..(x).........00000000
maxused..(x)............00079C00
limit..(x)..............02666680
failed..(x).............00000000
lock..(x)...............00000000
```

```
...

displaying kmemstats for offset 48 kalloc
address..................0025C7E0
inuse..(x)...............00000000
calls..(x)...............00000000
memuse..(x)..............00000000
limit blocks..(x)........00000000
map blocks..(x)..........00000000
maxused..(x).............00000000
limit..(x)...............02666680
failed..(x)..............00000000
lock..(x)................00000000

displaying kmemstats for offset 49 temp
address..................0025C804
inuse..(x)...............00000007
calls..(x)...............00000007
memuse..(x)..............00003500
(0)> more (^C to quit) ? //continue
limit blocks..(x)........00000000
map blocks..(x)..........00000000
maxused..(x).............00003500
limit..(x)...............02666680
failed..(x)..............00000000
lock..(x)................00000000
KDB(0)>
```

# Chapter 30. Display general and Journal File System (JFS) information subcommands

The subcommands in this category can be used to display general file system information, and information specific to the JFS filesystem. These subcommands include the following:

## dnlc subcommand

### Purpose

The **dnlc** subcommand displays information about the filesystem directory name lookup cache.

### Syntax

**dnlc** [*slot* | *effectiveaddress*]

### Parameters

- *slot* – Specifies the decimal identifier of a specific cache slot.
- *effectiveaddress* – Specifies the effective address of the entry. Symbols, hexadecimal values, or hexadecimal expressions can be used in specification of the address.

The **dnlc** subcommand is used to display information about the directory name cache.

When no parameters are provided, a summary of the entire directory name lookup cache is displayed.

### Aliases

**ncache**

### Example

The following is an example of how to use the **dnlc** subcommand:

```
KDB(0)> dnlc
                                        DP                NP NAME

      1 KERN_heap+59B9000 F10000F0049FBB48 F10000F004ED3D78 __vg10
      2 KERN_heap+59B9060 F10000F005009D78 F10000F00513FD78 CuAt.vc
      3 KERN_heap+59B90C0 F10000F0049FBB48 F10000F004E38D78 __pv16.0
      4 KERN_heap+59B9120 F10000F0049FBB48 F10000F0051DAD78 hd6
      5 KERN_heap+59B9180 F10000F0049FBB48 0000000000000000 __pv16.0
      6 KERN_heap+59B91E0 F10000F0049FBB48 F10000F004F6ED78 __pv16.0
      7 KERN_heap+59B9240 F10000F00557C918 F10000F005883918 libcrypt.a
      8 KERN_heap+59B92A0 F10000F0049FBB48 0000000000000000 __pv16.0
      9 KERN_heap+59B9300 F10000F0048C5B48 F10000F004B31B48 etc
     10 KERN_heap+59B9360 F10000F005009D78 F10000F0050A4D78 CuAt
     11 KERN_heap+59B93C0 F10000F004963D98 F10000F0051E1218 diagrpt23.dat
     12 KERN_heap+59B9420 F10000F0048C5B48 F10000F0049FBB48 dev
     13 KERN_heap+59B9480 F10000F004B31B48 F10000F004D02D78 vg
     14 KERN_heap+59B94E0 F10000F004B31B48 F10000F005009D78 objrepos
<snip>
KDB(0)> dnlc 14                                         //slot
                                        DP                NP NAME

     14 KERN_heap+59B94E0 F10000F004B31B48 F10000F005009D78 objrepos

vfsp..... F10000F0065F2470 forw..... F10000F0059B9300 back..... F10000F0059B9060

dp....... F10000F004B31B48 did...... 000000BA
np....... F10000F005009D78 nid...... 000000FF nidp..... F10000F005009E38
namelen.. 00000008
KDB(0)> dnlc F10000F0059B94E0    //eaddr
                                        DP                NP NAME

     14 KERN_heap+59B94E0 F10000F004B31B48 F10000F005009D78 objrepos

vfsp..... F10000F0065F2470 forw..... F10000F0059B9300 back..... F10000F0059B9060

dp....... F10000F004B31B48 did...... 000000BA
```

```
np....... F10000F005009D78 nid...... 000000FF nidp..... F10000F005009E38
namelen.. 00000008
KDB(0)> dnlc nlc_cache     //symbol
                                        DP                NP NAME

2146583247 nlc_cache+000000 0000000000020000 0000000000D22198

vfsp..... 0000000000020000 forw..... F10000F0059B9000 back..... 0000000000002000

dp....... 0000000000020000 did...... 00000000
np....... 0000000000D22198 nid...... FFFFFFFF nidp..... 0000000000D73F60
namelen.. FFFFFFFF
KDB(0)>
```

# hdnlc subcommand

## Purpose

The **hdnlc** subcommand displays information about the file system hash list for the directory name cache.

## Syntax

**hdnlc** [*slot* | *effectiveaddress*]

## Parameters

- *slot* – Specifies the decimal identifier of a specific hash bucket.
- *effectiveaddress* – Specifies the effective address of the entry. Symbols, hexadecimal values, or hexadecimal expressions can be used to specify the address.

The **hdnlc** command is used to display information about the dnlc hash table. When no parameters are provided, a summary of the entire hash list is displayed.

## Aliases

**hncache**

## Example

The following is an example of how to use the **hdnlc** subcommand:

```
KDB(0)> hdnlc
                BUCKET HEAD                     BACK      LOCK COUNT

KERN_heap+65B9000    1 F10000F0059B93C0 F10000F0059B9240 00000000    16
KERN_heap+65B9018    2 F10000F0059B9600 F10000F0059B9660 00000000     1
KERN_heap+65B9078    6 F10000F0059BAE60 F10000F0059BAEC0 00000000     2
KERN_heap+65B9288   28 F10000F0059C35C0 F10000F0059C3620 00000000    11
KERN_heap+65B9378   38 F10000F0059C6E00 F10000F0059C6E60 00000000     1
KERN_heap+65B9420   45 F10000F0059C9800 F10000F0059C9860 00000000     1
KERN_heap+65B9540   57 F10000F0059CE000 F10000F0059CE060 00000000     1
KERN_heap+65B9738   78 F10000F0059D5E00 F10000F0059D5E60 00000000     1
KERN_heap+65B9750   79 F10000F0059D6400 F10000F0059D6460 00000000     1
KERN_heap+65B9768   80 F10000F0059D6A00 F10000F0059D6A60 00000000     1
KERN_heap+65B9810   87 F10000F0059D9400 F10000F0059D9460 00000000     1
KERN_heap+65B9828   88 F10000F0059D9A00 F10000F0059D9A60 00000000     1
KERN_heap+65B98A0   93 F10000F0059DB800 F10000F0059DB860 00000000     1
KERN_heap+65B98D0   95 F10000F0059DC400 F10000F0059DC460 00000000     1
KERN_heap+65B9900   97 F10000F0059DD000 F10000F0059DD060 00000000     1
KERN_heap+65B9978  102 F10000F0059DEE00 F10000F0059DEE60 00000000     1
KERN_heap+65B9990  103 F10000F0059DF400 F10000F0059DF460 00000000     1
KERN_heap+65B9A38  110 F10000F0059E1E00 F10000F0059E1E60 00000000     1
KERN_heap+65B9A80  113 F10000F0059E3000 F10000F0059E3060 00000000     1
KERN_heap+65B9B88  124 F10000F0059E7200 F10000F0059E7260 00000000     1
(0)> more (^C to quit) ?
<snip>
KDB(0)> hdnlc 28           //specific bucket
HASH ENTRY( 28): F10000F0065B9288
                                        DP               NP NAME

    443 KERN_heap+59C35C0 F10000F0049FBB48 0000000000000000 __pv16.0
    442 KERN_heap+59C3560 F10000F0049FBB48 F10000F00557FFC8 __pv16.0
    441 KERN_heap+59C3500 F10000F0049FBB48 0000000000000000 __pv16.0
    440 KERN_heap+59C34A0 F10000F0049FBB48 F10000F0054E4FC8 __pv16.0
    439 KERN_heap+59C3440 F10000F0049FBB48 0000000000000000 __pv16.0
    438 KERN_heap+59C33E0 F10000F0049FBB48 F10000F00544A1F8 __pv16.0
    437 KERN_heap+59C3380 F10000F0049FBB48 0000000000000000 __pv16.0
    436 KERN_heap+59C3320 F10000F0049FBB48 F10000F0048C8B68 __pv16.0
```

```
    435 KERN_heap+59C32C0 F10000F0049FBB48 0000000000000000 __pv16.0
    434 KERN_heap+59C3260 F10000F0049FBB48 F10000F00557DA98 __pv16.0
    433 KERN_heap+59C3200 F10000F0049FBB48 0000000000000000 __pv16.0
    448 KERN_heap+59C37A0 0000000000000000 0000000000000000
    447 KERN_heap+59C3740 0000000000000000 0000000000000000
    446 KERN_heap+59C36E0 0000000000000000 0000000000000000
    445 KERN_heap+59C3680 0000000000000000 0000000000000000
    444 KERN_heap+59C3620 0000000000000000 0000000000000000
KDB(0)> hdnlc F10000F0065B9288          //effective address
HASH ENTRY( 28): F10000F0065B9288
                                        DP               NP NAME

    443 KERN_heap+59C35C0 F10000F0049FBB48 0000000000000000 __pv16.0
    442 KERN_heap+59C3560 F10000F0049FBB48 F10000F00557FFC8 __pv16.0
    441 KERN_heap+59C3500 F10000F0049FBB48 0000000000000000 __pv16.0
    440 KERN_heap+59C34A0 F10000F0049FBB48 F10000F0054E4FC8 __pv16.0
    439 KERN_heap+59C3440 F10000F0049FBB48 0000000000000000 __pv16.0
    438 KERN_heap+59C33E0 F10000F0049FBB48 F10000F00544A1F8 __pv16.0
    437 KERN_heap+59C3380 F10000F0049FBB48 0000000000000000 __pv16.0
    436 KERN_heap+59C3320 F10000F0049FBB48 F10000F0048C8B68 __pv16.0
    435 KERN_heap+59C32C0 F10000F0049FBB48 0000000000000000 __pv16.0
    434 KERN_heap+59C3260 F10000F0049FBB48 F10000F00557DA98 __pv16.0
    433 KERN_heap+59C3200 F10000F0049FBB48 0000000000000000 __pv16.0
    448 KERN_heap+59C37A0 0000000000000000 0000000000000000
    447 KERN_heap+59C3740 0000000000000000 0000000000000000
    446 KERN_heap+59C36E0 0000000000000000 0000000000000000
    445 KERN_heap+59C3680 0000000000000000 0000000000000000
    444 KERN_heap+59C3620 0000000000000000 0000000000000000
KDB(0)> hdnlc KERN_heap+65B9288    //effective address
HASH ENTRY( 28): F10000F0065B9288
                                        DP               NP NAME

    443 KERN_heap+59C35C0 F10000F0049FBB48 0000000000000000 __pv16.0
    442 KERN_heap+59C3560 F10000F0049FBB48 F10000F00557FFC8 __pv16.0
    441 KERN_heap+59C3500 F10000F0049FBB48 0000000000000000 __pv16.0
    440 KERN_heap+59C34A0 F10000F0049FBB48 F10000F0054E4FC8 __pv16.0
    439 KERN_heap+59C3440 F10000F0049FBB48 0000000000000000 __pv16.0
    438 KERN_heap+59C33E0 F10000F0049FBB48 F10000F00544A1F8 __pv16.0
    437 KERN_heap+59C3380 F10000F0049FBB48 0000000000000000 __pv16.0
    436 KERN_heap+59C3320 F10000F0049FBB48 F10000F0048C8B68 __pv16.0
    435 KERN_heap+59C32C0 F10000F0049FBB48 0000000000000000 __pv16.0
    434 KERN_heap+59C3260 F10000F0049FBB48 F10000F00557DA98 __pv16.0
    433 KERN_heap+59C3200 F10000F0049FBB48 0000000000000000 __pv16.0
    448 KERN_heap+59C37A0 0000000000000000 0000000000000000
    447 KERN_heap+59C3740 0000000000000000 0000000000000000
    446 KERN_heap+59C36E0 0000000000000000 0000000000000000
    445 KERN_heap+59C3680 0000000000000000 0000000000000000
    444 KERN_heap+59C3620 0000000000000000 0000000000000000
KDB(0)>
```

## kvn subcommand

### Purpose

The **kvn** subcommand displays the **kdm** vnode data structure.

### Syntax

**kvn** *address*

### Parameters

- *address* – Identifies the address of the **kdm** vnode to display.

### Aliases

No aliases.

### Example

The following is an example of how to use the **kvn** subcommand:

```
KDB(0)> kvn 0x3173F180
kdv_enables..0x00000000  kdv_flags....0x00000000  kdv_nreg.....0x00000000
kdv_op.......0x00801EC0  kdv_fset.....0x32F99400
kdv_regp.....0x00000000  kdv_data.....0x32F23628


NOTE: The kdm vnode pointer is in the JFS2 inode and may be obtained from
the output of the i2 command, in the kdmvp field:

KDB(0)> i2 32F23340
ADDRESS     DEVICE    I_NUM      IPMNT         COUNT   TYPE    FLAG
32F23340   002B0007     2       32F77020       00001   VDIR

In-memory Working Inode:
hashClass....0x000001B5  cacheClass...0x00000003  count........0x00000001
capability...0x000069C3  atlhead......0x00000000  atltail......0x00000000
bxflag.......0x00000000  blid.........0x00000000  btindex......0x00000000
diocnt.......0x00000001  nondiocnt....0x00000000
dev..........0x002B0007  synctime.....0x00000000  nodelock.....0x00000000
ipmnt........0x32F77020  ipimap.......0x32F13340  pagerObject..0x00000000
event........0xFFFFFFFF  fsevent......0xFFFFFFFF  openevent....0xFFFFFFFF
cacheLst.nxt.0x00000000  cacheLst.prv.0x00000000  freeNext.....0x00000000
hashLst.nxt..0x32F03340  hashLst.prv..0x31AA247C  kdmvp........0x3173F180
flag.........0x00000000                           ^^^^^^^^^^^^^^^^^^^^^^^
cflag........0x00000000
xlock........0x00000000
fsxlock......0x00000000
btorder......0x00000000
agstart......0x0000000000000000
lastCommittedSize...0x0000000000000100
```

## buffer subcommand

### Purpose

The **buffer** subcommand displays buffer cache headers.

### Syntax

**buffer** [*slot* | *effectiveaddress*]

### Parameters

- *slot* – Specifies the buffer pool slot number. This parameter must be a decimal value.
- *effectiveaddress* – Specifies the effective address of a buffer pool entry. Use symbols, hexadecimal values, or hexadecimal expressions to specify the address.

### Aliases

**buf**

### Example

The following is an example of how to use the **buffer** subcommand:

```
KDB(0)> buf  //print buffer pool
  1 057E4000 nodevice 00000000 00000000
  2 057E4058 nodevice 00000000 00000000
  3 057E40B0 nodevice 00000000 00000000
  4 057E4108 nodevice 00000000 00000000
  5 057E4160 nodevice 00000000 00000000
...
 18 057E45D8 nodevice 00000000 00000000
 19 057E4630 000A0011 00000000 00000100 READ DONE STALE MPSAFE MPSAFE_INITIAL
 20 057E4688 000A0011 00000000 00000008 READ DONE STALE MPSAFE MPSAFE_INITIAL
KDB(0) buf 19  //print buffer slot 19
                  DEV    VNODE    BLKNO FLAGS

 19 057E4630 000A0011 00000000 00000100 READ DONE STALE MPSAFE MPSAFE_INITIAL

forw     0562F0CC back     0562F0CC av_forw  057E45D8 av_back  057E4688
blkno    00000100 addr     0580C000 bcount   00001000 resid    00000000
error    00000000 work     80000000 options  00000000 event    FFFFFFFF
iodone:  biodone+000000
start.tv_sec       00000000 start.tv_nsec      00000000
xmemd.aspace_id    00000000 xmemd.xm_flag      00000000 xmemd.xm_version   00000000
xmemd.subspace_id  00000000 xmemd.subspace_id2 00000000 xmemd.uaddr        00000000
KDB(0)> pdt 17  //print paging device slot 17 (the 1st FS)

PDT address B69C0440 entry 17 of 511, type: FILESYSTEM
next pdt on i/o list  (nextio)  : FFFFFFFF
dev_t or strategy ptr (device)  : 000A0007
last frame w/pend I/O (iotail)  : FFFFFFFF
free buf_struct list  (bufstr)  : 056B2108
total buf structs     (nbufs)   : 005D
available (PAGING)    (avail)   : 0000
JFS  disk agsize      (agsize)  : 0800
JFS inode agsize      (iagsize) : 0800
JFS log SCB index     (logsidx) : 00035
JFS fragments per page(fperpage): 1
JFS compression type  (comptype): 0
JFS log2 bigalloc mult(bigexp)  : 0
disk map srval        (dmsrval) : 00002021
i/o's not finished    (iocnt)   : 00000000
lock                  (lock)    : E8003200
KDB(0)> buf 056B2108  //print paging device first free buffer
```

```
                 DEV    VNODE    BLKNO FLAGS

  0 056B2108 000A0007 00000000 00000048 DONE SPLIT MPSAFE MPSAFE_INITIAL

forw     0007DAB3 back     00000000 av_forw  056B20B0 av_back  00000000
blkno    00000048 addr     00000000 bcount   00001000 resid    00000000
error    00000000 work     00400000 options  00000000 event    00000000
iodone:  v_pfend+000000
start.tv_sec       00000000 start.tv_nsec      00000000
xmemd.aspace_id    00000000 xmemd.xm_flag      00000000 xmemd.xm_version   00000000
xmemd.subspace_id  0083E01F xmemd.subspace_id2 00000000 xmemd.uaddr        00000000
```

# hbuffer subcommand

## Purpose

The **hbuffer** subcommand displays buffer cache hash list headers.

## Syntax

**hbuffer** [*bucket* | *effectiveaddress*]

## Parameters

- *bucket* – Specifies the bucket number of the buffer cache hash list entry. This parameter must be a decimal value.
- *effectiveaddress* – Specifies the effective address of a buffer cache hash list entry. Use symbols, hexadecimal values, or hexadecimal expressions to specify the address.

If no parameter is specified, a summary for all entries is displayed. Display a specific entry by specifying the entry by bucket number or entry address.

## Aliases

**hb**

## Example

The following is an example of how to use the **hbuffer** subcommand:

```
KDB(0)> hb  //print buffer cache hash lists
         BUCKET HEAD     COUNT

0562F0CC   18   057E4630      1
0562F12C   26   057E4688      1
KDB(0)> hb 26  //print buffer cache hash list bucket 26
                  DEV    VNODE    BLKNO FLAGS

 20 057E4688 000A0011 00000000 00000008 READ DONE STALE MPSAFE MPSAFE_INITIAL
```

# fbuffer subcommand

## Purpose

The **fbuffer** subcommand displays buffer cache freelist headers.

## Syntax

**fbuffer** [*bucket* | *effectiveaddress*]

## Parameters

- *bucket* – Specifies the bucket number of the buffer cache freelist entry. This parameter must be a decimal value.
- *effectiveaddress* – Specifies the effective address of a buffer cache freelist entry. Use symbols, hexadecimal values, or hexadecimal expressions to specify the address.

If no parameter is specified, a summary for all entries is displayed. Display a specific entry by specifying the entry by bucket number or entry address.

## Aliases

**fb**

## Example

The following is an example of how to use the **fbuffer** subcommand:

```
KDB(0)> fb  //print free list buffer buckets
               BUCKET       HEAD  COUNT

bfreelist+000000 0001   057E4688     20
KDB(0)> fb 1  //print free list buffer bucket 1
                  DEV    VNODE    BLKNO FLAGS

 20 057E4688 000A0011 00000000 00000008 READ DONE STALE MPSAFE MPSAFE_INITIAL
 19 057E4630 000A0011 00000000 00000100 READ DONE STALE MPSAFE MPSAFE_INITIAL
 18 057E45D8 nodevice 00000000 00000000
 17 057E4580 nodevice 00000000 00000000
...
  2 057E4058 nodevice 00000000 00000000
  1 057E4000 nodevice 00000000 00000000
```

# gnode subcommand

## Purpose

The **gnode** subcommand displays the generic node structure at the specified address.

## Syntax

**gnode** *effectiveaddress*

## Parameters

- *effectiveaddress* – Specifies the effective address of a generic node structure. Use symbols, hexadecimal values, or hexadecimal expressions to specify the address.

## Aliases

**gno**

## Example

The following is an example of how to use the **gno** alias for the **gnode** subcommand:

```
(0)> gno 09D0FD68  //print gnode
GNODE............ 09D0FD68
gn_type....... 00000002 gn_flags...... 00000000 gn_seg........ 0001A3FA
gn_mwrcnt..... 00000000 gn_mrdcnt..... 00000000 gn_rdcnt...... 00000000
gn_wrcnt...... 00000000 gn_excnt...... 00000000 gn_rshcnt..... 00000000
gn_vnode...... 09D0FD28 gn_rdev....... 000A0010 gn_ops........ jfs_vops
gn_chan....... 00000000 gn_reclk_lock. 00000000 gn_reclk_lock@ 09D0FD9C
gn_reclk_event FFFFFFFF gn_filocks.... 00000000 gn_data....... 09D0FD58
gn_type....... DIR
```

## gfs subcommand

### Purpose

The **gfs** subcommand displays the generic file system structure at the specified address.

### Syntax

**gfs** *address*

### Parameters

- *address* - Specifies the address of a generic file system structure. Use symbols, hexadecimal values, or hexadecimal expressions to specify the address.

### Aliases

No aliases.

### Example

The following is an example of how to use the **gfs** subcommand:

```
(0)> gfs gfs  //print gfs slot 1
gfs_data. 00000000 gfs_flag. INIT VERSION4 VERSION42 VERSION421
gfs_ops.. jfs_vfsops     gn_ops... jfs_vops      gfs_name. jfs
gfs_init. jfs_init       gfs_rinit jfs_rootinit   gfs_type. JFS
gfs_hold. 00000012
(0)> gfs gfs+30  //print gfs slot 2
gfs_data. 00000000 gfs_flag. INIT VERSION4 VERSION42 VERSION421
gfs_ops.. spec_vfsops    gn_ops... spec_vnops     gfs_name. sfs
gfs_init. spec_init      gfs_rinit nodev          gfs_type. SFS
gfs_hold. 00000000
(0)> gfs gfs+60  //print gfs slot 3
gfs_data. 00000000 gfs_flag. REMOTE VERSION4
gfs_ops.. 01D2ABF8       gn_ops... 01D2A328       gfs_name. nfs
gfs_init. 01D2B5F0       gfs_rinit 00000000       gfs_type. NFS
gfs_hold. 0000000E
```

# file subcommand

## Purpose

The **file** subcommand displays file table entries.

## Syntax

**file** [ *slot* | *effectiveaddress*]

## Parameters

- *slot* – Specifies the slot number of a file table entry. This parameter must be a decimal value.
- *effectiveaddress* – Specifies the effective address of a file table entry. Use symbols, hexadecimal values, or hexadecimal expressions to specify the effective address.

If no parameter is used, all of the file table entries are displayed in a summary. Used files are displayed first. Detailed information can be displayed for individual file table entries by specifying the entry slot number or address.

## Aliases

No aliases.

## Example

The following is an example of how to use the **file** subcommand:

```
(0)> file  //print file table
                COUNT           OFFSET     DATA TYPE   FLAGS

   1 file+000000     1 0000000000000100 09CD90C8 VNODE  EXEC
   2 file+000030     1 0000000000000100 09CC4DE8 VNODE  EXEC
   3 file+000060  1452 000000000019B084 09CC2B50 VNODE  READ RSHARE
   4 file+000090     2 0000000000000100 09CFCD80 VNODE  EXEC
   5 file+0000C0     2 0000000000000000 056CE008 VNODE  READ WRITE
   6 file+0000F0     1 0000000000000000 056CE008 VNODE  READ WRITE
   7 file+000120     1 0000000000000680 09CFF680 VNODE  READ WRITE
   8 file+000150     1 0000000000000100 0B97BE0C VNODE  EXEC
   9 file+000180     2 0000000000000000 056CE070 VNODE  READ NONBLOCK
  10 file+0001B0   323 000000000000061C 09CC4F30 VNODE  READ RSHARE
  11 file+0001E0     1 0000000000000000 0B7E8700 READ WRITE
  12 file+000210    16 000000000000061C 09CC5AB8 VNODE  READ RSHARE
  13 file+000240     1 0000000000000000 0B221950 GNODE  WRITE
  14 file+000270     1 0000000000000000 0B221A20 GNODE  WRITE
  15 file+0002A0     2 000000000000055C 09CFFCE8 VNODE  READ RSHARE
  16 file+0002D0     2 0000000000000000 09CFE9B0 VNODE  WRITE
  17 file+000300     1 0000000000000000 0B7E8600 READ WRITE
  18 file+000330     1 0000000000000000 056CE008 VNODE  READ
  19 file+000360     1 0000000000000000 09CFBB90 VNODE  WRITE
  20 file+000390     3 000000000000284A 0B99A60C VNODE  READ
(0)> more (^C to quit) ? Interrupted
(0)> file 3  //print file slot 3
                COUNT           OFFSET     DATA TYPE   FLAGS

   3 file+000060  1474 000000000019B084 09CC2B50 VNODE  READ RSHARE

f_flag......... 00001001 f_count........ 000005C2
f_msgcount......... 0000 f_type............. 0001
f_data......... 09CC2B50 f_offset... 000000000019B084
f_dir_off...... 00000000 f_cred......... 056D0E58
f_lock@........ 004AF098 f_lock......... 00000000
f_offset_lock@. 004AF09C f_offset_lock.. 00000000
f_vinfo........ 00000000 f_ops.......... 00250FC0 vnodefops+000000
```

```
VNODE.......... 09CC2B50
v_flag.... 00000000 v_count... 00000002 v_vfsgen.. 00000000
v_lock.... 00000000 v_lock@... 09CC2B5C v_vfsp.... 056D18A4
v_mvfsp... 00000000 v_gnode... 09CC2B90 v_next.... 00000000
v_vfsnext. 09CC2A08 v_vfsprev. 09CC3968 v_pfsvnode 00000000
v_audit... 00000000
```

# inode subcommand

## Purpose

The **inode** subcommand displays inode table entries.

## Syntax

**inode** [*slot* | *effectiveaddress*]

## Parameters

- *slot* – Specifies the slot number of an inode table entry. This parameter must be a decimal value.
- *effectiveaddress* – Specifies the effective address of an inode table entry. Use symbols, hexadecimal values, or hexadecimal expressions to specify the address.

If no parameter is entered, a summary for used inode table entries is displayed. The inode is considered used when count is greater than 0. Unused inodes are displayed with the **fino** subcommand. Detailed information is displayed for individual inode table entries by specifying the entry. The information is interpreted for special inodes. Special inodes include: mountnode and inodes.

## Aliases

**ino**

## Example

The following is an example of how to use the **ino** alias for the **inode** subcommand:

```
(0)> ino  //print inode table
                   DEV     NUMBER CNT    GNODE    IPMNT TYPE FLAGS

   1 0A2A4968 00330003      10721   1 0A2A4978 09F79510 DIR
   2 0A2A9790 00330003      10730   1 0A2A97A0 09F79510 REG
   3 0A321E90 00330006       2948   1 0A321EA0 09F7A990 DIR
   4 0A32ECD8 00330006       2965   1 0A32ECE8 09F7A990 DIR
   5 0A38EBC8 00330006       3173   1 0A38EBD8 09F7A990 DIR
   6 0A3CC280 00330006       3186   1 0A3CC290 09F7A990 REG
   7 09D01570 000A0005      14417   1 09D01580 09CC1990 REG
   8 09D7CE68 000A0005      47211   1 09D7CE78 09CC1990 REG  ACC
   9 09D1A530 000A0005       6543   1 09D1A540 09CC1990 REG
  10 09D19C38 000A0005       6542   1 09D19C48 09CC1990 REG
  11 09CFFD18 000A0005      71811   1 09CFFD28 09CC1990 REG
  12 09D00238 000A0005      63718   1 09D00248 09CC1990 REG
  13 09D70918 000A0005       6746   1 09D70928 09CC1990 REG
  14 09D01800 000A0005      15184   1 09D01810 09CC1990 REG
  15 09F9B450 00330003       4098   1 09F9B460 09F79510 DIR
  16 09F996D8 00330003       4097   1 09F996E8 09F79510 DIR
  17 0A5C6548 00330006       4110   1 0A5C6558 09F7A990 DIR
  18 09FB30D8 00330005       4104   1 09FB30E8 09F79F50 DIR  CHG UPD FSYNC DIRTY
  19 09FAB868 00330003       4117   1 09FAB878 09F79510 REG
  20 0A492AB8 00330003       4123   1 0A492AC8 09F79510 REG
(0)> more (^C to quit) ?  //Interrupted
(0)> ino 09F79510  //print mount table inode (IPMNT)
                   DEV     NUMBER CNT    GNODE    IPMNT TYPE FLAGS

     09F79510 00330003          0   1 09F79520 09F79510 NON  CMNEW


forw      09F78C18 back      09F7A5B8 next      09F79510 prev      09F79510
gnode@    09F79520 number    00000000 dev       00330003 ipmnt     09F79510
flag      00000000 locks     00000000 bigexp    00000000 compress  00000000
cflag     00000002 count     00000001 event     FFFFFFFF movedfrag 00000000
openevent FFFFFFFF id        000052AB hip       09C9C330 nodelock  00000000
```

```
nodelock@ 09F79590 dquot[USR]00000000 dquot[GRP]00000000 dinode@   09F7959C
cluster   00000000 size      0000000000000000

GNODE............ 09F79520
gn_type....... 00000000 gn_flags...... 00000000 gn_seg........ 00000000
gn_mwrcnt..... 00000000 gn_mrdcnt..... 00000000 gn_rdcnt...... 00000000
gn_wrcnt...... 00000000 gn_excnt...... 00000000 gn_rshcnt..... 00000000
gn_vnode...... 09F794E0 gn_rdev....... 00000000 gn_ops........ jfs_vops
gn_chan....... 00000000 gn_reclk_lock. 00000000 gn_reclk_lock@ 09F79554
gn_reclk_event FFFFFFFF gn_filocks.... 00000000 gn_data....... 09F79510
gn_type....... NON

di_gen        32B69977 di_mode       00000000 di_nlink      00000000
di_acct       00000000 di_uid        00000000 di_gid        00000000
di_nblocks    00000000 di_acl        00000000
di_mtime      00000000 di_atime      00000000 di_ctime      00000000
di_size_hi    00000000 di_size_lo    00000000

VNODE........... 09F794E0
v_flag.... 00000000 v_count... 00000000 v_vfsgen.. 00000000
v_lock.... 00000000 v_lock@... 09F794EC v_vfsp.... 00000000
v_mvfsp... 00000000 v_gnode... 09F79520 v_next.... 00000000
v_vfsnext. 00000000 v_vfsprev. 00000000 v_pfsvnode 00000000
v_audit... 00000000

di_iplog      09F77F48 di_ipinode    09F798E8 di_ipind      09F797A0
di_ipinomap   09F79A30 di_ipdmap     09F79B78 di_ipsuper    09F79658
di_ipinodex   09F79CC0 di_jmpmnt     0B8E0B00
di_agsize     00004000 di_iagsize    00000800 di_logsidx    00000547
di_fperpage   00000008 di_fsbigexp   00000000 di_fscompress 00000001

(0)> ino 09F77F48  //print log inode (di_iplog)
                   DEV     NUMBER CNT    GNODE    IPMNT TYPE FLAGS

     09F77F48 00330001          0   5 09F77F58 09F77F48 NON  CMNEW


forw      09C9C310 back      09F785B0 next      09F77F48 prev      09F77F48
gnode@    09F77F58 number    00000000 dev       00330001 ipmnt     09F77F48
flag      00000000 locks     00000000 bigexp    00000000 compress  00000000
cflag     00000002 count     00000005 event     FFFFFFFF movedfrag 00000000
openevent FFFFFFFF id        0000529A hip       09C9C310 nodelock  00000000
nodelock@ 09F77FC8 dquot[USR]00000000 dquot[GRP]00000000 dinode@   09F77FD4
cluster   00000000 size      0000000000000000

GNODE............ 09F77F58
gn_type....... 00000000 gn_flags...... 00000000 gn_seg........ 00007547
gn_mwrcnt..... 00000000 gn_mrdcnt..... 00000000 gn_rdcnt...... 00000000
gn_wrcnt...... 00000000 gn_excnt...... 00000000 gn_rshcnt..... 00000000
gn_vnode...... 09F77F18 gn_rdev....... 00000000 gn_ops........ jfs_vops
gn_chan....... 00000000 gn_reclk_lock. 00000000 gn_reclk_lock@ 09F77F8C
gn_reclk_event FFFFFFFF gn_filocks.... 00000000 gn_data....... 09F77F48
gn_type....... NON

di_gen        32B69976 di_mode       00000000 di_nlink      00000000
di_acct       00000000 di_uid        00000000 di_gid        00000000
di_nblocks    00000000 di_acl        00000000
di_mtime      00000000 di_atime      00000000 di_ctime      00000000
di_size_hi    00000000 di_size_lo    00000000

VNODE........... 09F77F18
v_flag.... 00000000 v_count... 00000000 v_vfsgen.. 00000000
v_lock.... 00000000 v_lock@... 09F77F24 v_vfsp.... 00000000
v_mvfsp... 00000000 v_gnode... 09F77F58 v_next.... 00000000
v_vfsnext. 00000000 v_vfsprev. 00000000 v_pfsvnode 00000000
v_audit... 00000000
```

```
di_logptr    0000015A di_logsize   00000C00 di_logend    00000FF8
di_logsync   0005A994 di_nextsync  0013BBFC di_logxor    6C868513
di_llogeor   00000FE0 di_llogxor   6CE29103 di_logx      0BB13200
di_logdgp    0B7E5BC0 di_loglock   4004B9EF di_loglock@  09F7804C
logxlock     00000000 logxlock@    0BB13200 logflag      00000001
logppong     00000195 logcq.head   B69CAB7C logcq.tail   0BB13228
logcsn       00001534 logcrtc      0000000C loglcrt      B69CA97C
logeopm      00000001 logeopmc     00000002
logeopmq[0]@ 0BB13228 logeopmq[1]@ 0BB13268
```

# hinode subcommand

## Purpose

The **hinode** subcommand displays inode hash list entries.

## Syntax

**hinode** [*bucket* | *address*]

## Parameters

- *bucket* – Specifies the bucket number. This parameter must be a decimal value.
- *address* – Specifies the effective address of an inode hash list entry. Use symbols, hexadecimal values, or hexadecimal expressions to specify the address.

If no parameter is entered, the hash list is displayed. View the entries for a specific hash table entry by specifying a bucket number or the address of a hash list bucket.

## Aliases

**hino**

## Example

The following is an example of how to use the **hino** alias for the **hinode** subcommand:

```
(0)> hino  //print hash inode buckets
         BUCKET HEAD    TIMESTAMP     LOCK COUNT

09C86000    1   0A285470 00000005 00000000     4
09C86010    2   0A284E08 00000006 00000000     3
09C86020    3   0A2843C8 00000006 00000000     3
09C86030    4   0A287EB8 00000006 00000000     3
09C86040    5   0A287330 00000005 00000000     3
09C86050    6   0A2867A8 00000006 00000000     4
09C86060    7   0A285FF8 00000007 00000000     3
09C86070    8   0A289D78 00000006 00000000     4
09C86080    9   0A289858 00000006 00000000     4
09C86090   10   0A33E2D8 00000005 00000000     4
09C860A0   11   0A33E7F8 00000005 00000000     4
09C860B0   12   0A33EE60 00000005 00000000     4
09C860C0   13   0A33F758 00000005 00000000     4
09C860D0   14   0A28AE20 00000005 00000000     3
09C860E0   15   0A28A670 00000005 00000000     3
09C860F0   16   0A33CE58 00000005 00000000     4
09C86100   17   0A33D9E0 00000006 00000000     4
09C86110   18   0A5FF6D0 00000008 00000000     4
09C86120   19   0A5FD060 00000009 00000000     4
09C86130   20   0A5FC390 00000009 00000000     4
(0)> more (^C to quit) ? Interrupted
(0)> hino 18  //print hash inode bucket 18
HASH ENTRY( 18): 09C86110
                   DEV     NUMBER CNT    GNODE    IPMNT TYPE FLAGS

     0A5FF6D0 00330003       2523   0 0A5FF6E0 09F79510 REG
     0A340E68 00330004       2524   0 0A340E78 09F78090 REG
     0A28CA50 00330003      10677   0 0A28CA60 09F79510 DIR
     0A1AFCA0 00330006       2526   0 0A1AFCB0 09F7A990 REG
```

## icache subcommand

### Purpose

The **icache** subcommand displays inode cache list entries.

### Syntax

**icache** [*slot* | *effectiveaddress*]

### Parameters

- *slot* – Specifies the slot number of an inode cache list entry. This parameter must be a decimal value.
- *effectiveaddress* – Specifies the effective address of an inode cache list entry. Use symbols, hexadecimal values, or hexadecimal expressions to specify the address.

If no parameter is entered, a summary is displayed. Display detailed information for a particular entry by specifying the entry to display with either the slot number or the address.

### Aliases

**fino**

### Example

The following is an example of how to use the **fino** alias for the **icache** subcommand:

```
(0)> fino  //print free inode cache
                  DEV     NUMBER CNT    GNODE    IPMNT TYPE FLAGS

   1 09CABFA0 DEADBEEF         0   0 09CABFB0 09CA7178 CHR  CMNOLINK
   2 0A8D3A70 DEADBEEF         0   0 0A8D3A80 09F7A990 REG  CMNOLINK
   3 0A8F2528 DEADBEEF         0   0 0A8F2538 09CC6528 REG  CMNOLINK
   4 0A7C66E0 DEADBEEF         0   0 0A7C66F0 09F7A990 REG  CMNOLINK
   5 0A7BA568 DEADBEEF         0   0 0A7BA578 09F79F50 REG  CMNOLINK
   6 0A78EC68 DEADBEEF         0   0 0A78EC78 09F78090 REG  CMNOLINK
   7 0A7AF9B8 DEADBEEF         0   0 0A7AF9C8 09F79F50 REG  CMNOLINK
   8 0A7B9230 DEADBEEF         0   0 0A7B9240 09F79F50 REG  CMNOLINK
   9 0A8BDCA8 DEADBEEF         0   0 0A8BDCB8 09F79F50 LNK  CMNOLINK
  10 0A8BE978 DEADBEEF         0   0 0A8BE988 09F7A990 REG  CMNOLINK
  11 0A7C58C8 DEADBEEF         0   0 0A7C58D8 09F7A990 REG  CMNOLINK
  12 0A78D6A0 DEADBEEF         0   0 0A78D6B0 09F78090 REG  CMNOLINK
  13 0A7C4BF8 DEADBEEF         0   0 0A7C4C08 09F7A990 REG  CMNOLINK
  14 0A78ADA0 DEADBEEF         0   0 0A78ADB0 09F78090 REG  CMNOLINK
  15 0A7B8A80 DEADBEEF         0   0 0A7B8A90 09F79F50 REG  CMNOLINK
  16 0A8BC970 DEADBEEF         0   0 0A8BC980 09F7A990 REG  CMNOLINK
  17 0A8D1CF8 DEADBEEF         0   0 0A8D1D08 09F7A990 REG  CMNOLINK
  18 0A7AE160 DEADBEEF         0   0 0A7AE170 09F79F50 REG  CMNOLINK
  19 0A8EF998 DEADBEEF         0   0 0A8EF9A8 09CC6528 REG  CMNOLINK
  20 0A7C41B8 DEADBEEF         0   0 0A7C41C8 09F7A990 REG  CMNOLINK
(0)> more (^C to quit) ? Interrupted
(0)> fino 1  //print free inode slot 1
                  DEV     NUMBER CNT    GNODE    IPMNT TYPE FLAGS

     09CABFA0 DEADBEEF         0   0 09CABFB0 09CA7178 CHR  CMNOLINK


forw      09CABFA0 back      09CABFA0 next      0A8EF708 prev      0042AE60
gnode@    09CABFB0 number    00000000 dev       DEADBEEF ipmnt     09CA7178
flag      00000000 locks     00000000 bigexp    00000000 compress  00000000
cflag     00000004 count     00000000 event     FFFFFFFF movedfrag 00000000
openevent FFFFFFFF id        00000045 hip       00000000 nodelock  00000000
nodelock@ 09CAC020 dquot[USR]00000000 dquot[GRP]00000000 dinode@   09CAC02C
cluster   00000000 size      0000000000000000
```

```
GNODE............ 09CABFB0
gn_type....... 00000004 gn_flags...... 00000000 gn_seg........ 00000000
gn_mwrcnt..... 00000000 gn_mrdcnt..... 00000000 gn_rdcnt...... 00000000
gn_wrcnt...... 00000000 gn_excnt...... 00000000 gn_rshcnt..... 00000000
gn_vnode...... 09CABF70 gn_rdev....... 00030000 gn_ops........ jfs_vops
gn_chan....... 00000000 gn_reclk_lock. 00000000 gn_reclk_lock@ 09CABFE4
gn_reclk_event FFFFFFFF gn_filocks.... 00000000 gn_data....... 09CABFA0
gn_type....... CHR

di_gen        00000000 di_mode       00000000 di_nlink      00000000
di_acct       00000000 di_uid        00000000 di_gid        00000000
di_nblocks    00000000 di_acl        00000000
di_mtime      32B67A97 di_atime      32B67A97 di_ctime      32B67B4B
di_size_hi    00000000 di_size_lo    00000000
di_rdev       00030000

VNODE........... 09CABF70
v_flag.... 00000000 v_count... 00000000 v_vfsgen.. 00000000
v_lock.... 00000000 v_lock@... 09CABF7C v_vfsp.... 00000000
v_mvfsp... 00000000 v_gnode... 09CABFB0 v_next.... 00000000
v_vfsnext. 09CABE28 v_vfsprev. 00000000 v_pfsvnode 00000000
v_audit... 00000000
```

## vnc subcommand

### Purpose

The **vnc** subcommand displays information about the vnode cache filesystem.

### Syntax

**vnc** [*slot* | *effectiveaddress*]

### Parameters

- *slot* – Specifies the decimal identifier of a specific cache entry.
- *effectiveaddress* – Specifies the effective address of the entry.

You can only specify one parameter.

When no parameters are provided, a summary of the entire **vnode** cache is displayed. If there are no valid cache entries in memory, nothing is displayed.

### Aliases

**vcache**

### Example

The following is an example of how to use the **vnc** subcommand:

```
(0)> vnc
                                     VP              DEV      INO

   1 F10010F0109A0000 F10010F007F929B0 8000000A0000000B 00000000
   2 F10010F0109A0028 F10010F0109A0060 0000000000000000 80000021
   3 F10010F0109A0050 F10010F0109A0030 F10010F0109A0090 00000000
   8 F10010F0109A0118 F10010F0109A0150 0000000000000000 00000000
   9 F10010F0109A0140 F10010F0109A0120 F10010F0109A0180 00000000
  14 F10010F0109A0208 F10010F0109A0240 0000000000000000 00000000
  15 F10010F0109A0230 F10010F0109A0210 F10010F0109A0270 00000000
  20 F10010F0109A02F8 F10010F0109A0330 0000000000000000 00000000
  21 F10010F0109A0320 F10010F0109A0300 F10010F0109A0360 00000000
  26 F10010F0109A03E8 F10010F0109A0420 0000000000000000 00000000
  27 F10010F0109A0410 F10010F0109A03F0 F10010F0109A0450 00000000
  32 F10010F0109A04D8 F10010F0109A0510 0000000000000000 00000000
  33 F10010F0109A0500 F10010F0109A04E0 F10010F0109A0540 00000000
  38 F10010F0109A05C8 F10010F0109A0000 0000000000000000 00000000
  39 F10010F0109A05F0 F10010F0109A0BD0 F10010F0121A0018 F10010F0
  40 F10010F0109A0618 68E7612700000000 F10010F0121A0018 F10010F0
  44 F10010F0109A06B8 F10010F0109A06F0 0000000000000000 00000000
  45 F10010F0109A06E0 F10010F0109A06C0 F10010F0109A0720 00000000
  50 F10010F0109A07A8 F10010F0109A07E0 0000000000000000 00000000
  51 F10010F0109A07D0 F10010F0109A07B0 F10010F0109A0810 00000000
(0)> more (^C to quit) ? //Interrupted

(0)> vnc 1
                                     VP              DEV      INO

   1 F10010F0109A0000 F10010F007F929B0 8000000A0000000B 00000000
forw..... F10010F0109A05D0 back..... F10010F0121A0000
ino...... 00000000 gen...... 0000C125
v_flag.... 00000000 v_count... 00000001
v_vfsgen.. 00000000 v_vfsp.... F10010F00EE42940
v_lock@... F10010F007F929C0 v_lock.... 0000000000000000
```

```
v_mvfsp... 0000000000000000 v_gnode... F10010F007F92A28
v_next.... 0000000000000000 v_vfsnext. F10010F007D629B0
v_vfsprev. F10010F0081C29B0 v_pfsvnode 0000000000000000
v_audit... 0000000000000000
```

## hvnc subcommand

### Purpose

The **hvnc** subcommand displays information about the filesystem hash list for the vnode cache.

### Syntax

**hvnc** [*slot* | *effectiveaddress*]

### Parameters

- *slot* – Specifies the decimal identifier of a specific hash bucket.
- *effectiveaddress* – Specifies the effective address of the entry. Symbols, hexadecimal values, or hexadecimal expressions can be used in specification of the address.

The **hvnc** command is used to display information about the **vcache** hash table. When no parameters are provided, a summary of the entire hash list is displayed.

### Aliases

**hvcache**

### Example

The following is an example of how to use the **hvnc** subcommand:

```
(0)> hvnc
                BUCKET HEAD                         BACK     LOCK COUNT

F10010F0121A0000    1 F10010F0109A0000 F10010F0109A0030 00000000     1
F10010F0121A0018    2 F10010F0109A0600 F10010F0109A0630 00000000     1
F10010F0121A0060    5 F10010F0109A1800 F10010F0109A1830 00000000     1
F10010F0121A0078    6 F10010F0109A1E00 F10010F0109A1E30 00000000     1
F10010F0121A00C0    9 F10010F0109A3000 F10010F0109A3030 00000000     1
F10010F0121A00F0   11 F10010F0109A3C00 F10010F0109A3C30 00000000     1
F10010F0121A0108   12 F10010F0109A4200 F10010F0109A4230 00000000     1
F10010F0121A0138   14 F10010F0109A4E00 F10010F0109A4E30 00000000     1
F10010F0121A0150   15 F10010F0109A5400 F10010F0109A5430 00000000     1
F10010F0121A0168   16 F10010F0109A5A00 F10010F0109A5A30 00000000     1
F10010F0121A01B0   19 F10010F0109A6C00 F10010F0109A6C30 00000000     1
F10010F0121A01C8   20 F10010F0109A7230 F10010F0109A7260 00000000     2
F10010F0121A01E0   21 F10010F0109A7800 F10010F0109A7830 00000000     1
F10010F0121A01F8   22 F10010F0109A7E00 F10010F0109A7E30 00000000     1
F10010F0121A0228   24 F10010F0109A8A00 F10010F0109A8A30 00000000     1
F10010F0121A0240   25 F10010F0109A9060 F10010F0109A9090 00000000     3
F10010F0121A0258   26 F10010F0109A9600 F10010F0109A9630 00000000     1
F10010F0121A0270   27 F10010F0109A9C00 F10010F0109A9C30 00000000     1
F10010F0121A02B8   30 F10010F0109AAE30 F10010F0109AAE60 00000000     2
F10010F0121A02D0   31 F10010F0109AB400 F10010F0109AB430 00000000     1
(0)> more (^C to quit) ? //Interrupted

(0)> hvnc 1
HASH ENTRY(  1): F10010F0121A0000
                                     VP              DEV      INO

   1 F10010F0109A0000 F10010F007F929B0 8000000A0000000B 00000000
  38 F10010F0109A05D0 0000000000000000 0000000000000000 00000000
  37 F10010F0109A05A0 0000000000000000 0000000000000000 00000000
  35 F10010F0109A0570 0000000000000000 0000000000000000 00000000
  34 F10010F0109A0540 0000000000000000 0000000000000000 00000000
  33 F10010F0109A0510 0000000000000000 0000000000000000 00000000
  32 F10010F0109A04E0 0000000000000000 0000000000000000 00000000
  31 F10010F0109A04B0 0000000000000000 0000000000000000 00000000
```

```
  29 F10010F0109A0480 0000000000000000 0000000000000000 00000000
  28 F10010F0109A0450 0000000000000000 0000000000000000 00000000
  27 F10010F0109A0420 0000000000000000 0000000000000000 00000000
  26 F10010F0109A03F0 0000000000000000 0000000000000000 00000000
  25 F10010F0109A03C0 0000000000000000 0000000000000000 00000000
  23 F10010F0109A0390 0000000000000000 0000000000000000 00000000
  22 F10010F0109A0360 0000000000000000 0000000000000000 00000000
  21 F10010F0109A0330 0000000000000000 0000000000000000 00000000
  20 F10010F0109A0300 0000000000000000 0000000000000000 00000000
  19 F10010F0109A02D0 0000000000000000 0000000000000000 00000000
  17 F10010F0109A02A0 0000000000000000 0000000000000000 00000000
  16 F10010F0109A0270 0000000000000000 0000000000000000 00000000
(0)>
```

## vnode subcommand

### Purpose

The **vnode** subcommand displays virtual node (vnode) table entries.

### Syntax

**vnode** [*slot* | *effectiveaddress*]

### Parameters

- *slot* – Specifies the slot number of a virtual node table entry. This parameter must be a decimal value.
- *effectiveaddress* – Specifies the effective address of a virtual node table entry. Use symbols, hexadecimal values, or hexadecimal expressions to specify the address.

If no parameter is entered, a summary is displayed with one line per table entry. Display detailed information for individual vnode table entries by specifying the entry with either a slot number or an address.

### Aliases

**vno**

### Example

The following is an example of how to use the **vnode** subcommand:

```
(0)> vnode  //print vnode table
              COUNT VFSGEN    GNODE     VFSP  DATAPTR TYPE FLAGS

 106 09D227B0     3      0 09D227F0 056D183C 00000000 REG
 126 09D1AB68     1      0 09D1ABA8 056D183C 00000000 REG
 130 09D196E8     1      0 09D19728 056D183C 00000000 REG
 135 09D18B60     1      0 09D18BA0 056D183C 05CC2D00 SOCK
 140 09D17E90     1      0 09D17ED0 056D183C 05D3F300 SOCK
 143 09D17970     1      0 09D179B0 056D183C 05CC2A00 SOCK
 148 09D17078     1      0 09D170B8 056D183C 05CC2800 SOCK
 154 09D14DE0     1      0 09D14E20 056D183C 00000000 REG
 162 09D13818     1      0 09D13858 056D183C 05D30E00 SOCK
 165 09D0D948     1      0 09D0D988 056D183C 00000000 DIR
 166 09D0D800     1      0 09D0D840 056D183C 00000000 DIR
 167 09D0D6B8     1      0 09D0D6F8 056D183C 00000000 DIR
 168 09D0D570     1      0 09D0D5B0 056D183C 00000000 DIR
 170 09D0D2E0     1      0 09D0D320 056D183C 00000000 DIR
 171 09D0D198     1      0 09D0D1D8 056D183C 00000000 DIR
 172 09D0D050     1      0 09D0D090 056D183C 00000000 DIR
 173 09D0CF08     1      0 09D0CF48 056D183C 00000000 DIR
 174 09D0CDC0     1      0 09D0CE00 056D183C 00000000 DIR
 175 09D0CC78     1      0 09D0CCB8 056D183C 00000000 DIR
 176 09D0CB30     1      0 09D0CB70 056D183C 00000000 DIR
(0)> more (^C to quit) ?  //Interrupted
(0)> vnode 106  //print vnode slot 106
              COUNT VFSGEN    GNODE     VFSP  DATAPTR TYPE FLAGS

 106 09D227B0     3      0 09D227F0 056D183C 00000000 REG
v_flag.... 00000000 v_count... 00000003 v_vfsgen.. 00000000
v_lock.... 00000000 v_lock@... 09D227BC v_vfsp.... 056D183C
v_mvfsp... 00000000 v_gnode... 09D227F0 v_next.... 00000000
v_vfsnext. 09D22668 v_vfsprev. 09D22B88 v_pfsvnode 00000000
v_audit... 00000000
```

# vfs subcommand

## Purpose

The **vfs** subcommand displays entries of the virtual file system table.

## Syntax

**vfs** [*slot* | *address*]

## Parameters

- *slot* – Specifies the slot number of a virtual file system table entry. This parameter must be a decimal value.
- *address* – Specifies the address of a virtual file system table entry. Use symbols, hexadecimal values, or hexadecimal expressions to specify the address.

If no parameter is entered, a summary is displayed with one line for each entry. Display detailed information by identifying the entry of interest with either a slot number or an address.

## Aliases

**mount**

## Example

The following is an example of how to use the **vfs** subcommand:

```
 (0)> vfs  //print vfs table
                 GFS     MNTD MNTDOVER   VNODES     DATA TYPE   FLAGS

  1 056D183C 0024F268 09CC08B8 00000000 0A5AADA0 0B221F68 JFS    DEVMOUNT
... /dev/hd4 mounted over /
  2 056D18A4 0024F268 09CC2258 09CC0B48 0A545270 0B221F00 JFS    DEVMOUNT
... /dev/hd2 mounted over /usr
  3 056D1870 0024F268 09CC3820 09CC2DE0 09D913A8 0B221E30 JFS    DEVMOUNT
... /dev/hd9var mounted over /var
  4 056D1808 0024F268 09CC6DF0 09CC6120 0A7DC1E8 0B221818 JFS    DEVMOUNT
... /dev/hd3 mounted over /tmp
  5 056D18D8 0024F268 09D0BFA8 09D0B568 09D95500 0B2412F0 JFS    DEVMOUNT
... /dev/hd1 mounted over /home
  6 056D190C 0024F2C8 0B243C0C 09D0C238 0B9F6A0C 0B230500 NFS    READONLY REMOTE
... /pvt/tools mounted over /pvt/tools
  7 056D1940 0024F2C8 0B7E440C 09D0CB30 0B985C0C 0B230A00 NFS    READONLY REMOTE
... /pvt/base mounted over /pvt/base
  8 056D1974 0024F2C8 0B7E4A0C 09D0CC78 0B7E4A0C 0B230C00 NFS    READONLY REMOTE
... /pvt/periph mounted over /pvt/periph
  9 056D19A8 0024F2C8 0B7E4E0C 09D0CDC0 0B89000C 0B230E00 NFS    READONLY REMOTE
... /nfs mounted over /nfs
 10 056D19DC 0024F2C8 0B89020C 09D0CF08 0B89840C 0B230000 NFS    READONLY REMOTE
... /tcp mounted over /tcp
(0)> vfs 5  //print vfs slot 5
                 GFS     MNTD MNTDOVER   VNODES     DATA TYPE   FLAGS

  5 056D18D8 0024F268 09D0BFA8 09D0B568 09D95500 0B2412F0 JFS    DEVMOUNT
... /dev/hd1 mounted over /home

vfs_next..... 056D190C vfs_count.... 00000001 vfs_mntd..... 09D0BFA8
vfs_mntdover. 09D0B568 vfs_vnodes... 09D95500 vfs_count.... 00000001
vfs_number... 00000009 vfs_bsize.... 00001000 vfs_mdata.... 0B7E8E80
vmt_revision. 00000001 vmt_length... 00000070 vfs_fsid..... 000A0008 00000003
vmt_vfsnumber 00000009 vfs_date..... 32B67BFF vfs_flag..... 00000004
vmt_gfstype.. 00000003 @vmt_data.... 0B7E8EA4 vfs_lock..... 00000000
vfs_lock@.... 056D1904 vfs_type..... 00000003 vfs_ops...... jfs_vfsops
```

```
VFS_GFS.. gfs+000000
gfs_data. 00000000 gfs_flag. INIT VERSION4 VERSION42 VERSION421
gfs_ops.. jfs_vfsops     gn_ops... jfs_vops       gfs_name. jfs
gfs_init. jfs_init       gfs_rinit jfs_rootinit   gfs_type. JFS
gfs_hold. 00000013

VFS_MNTD.. 09D0BFA8
v_flag.... 00000001 v_count... 00000001 v_vfsgen.. 00000000
v_lock.... 00000000 v_lock@... 09D0BFB4 v_vfsp.... 056D18D8
v_mvfsp... 00000000 v_gnode... 09D0BFE8 v_next.... 00000000
v_vfsnext. 00000000 v_vfsprev. 09D730A0 v_pfsvnode 00000000
v_audit... 00000000 v_flag.... ROOT

VFS_MNTDOVER.. 09D0B568
v_flag.... 00000000 v_count... 00000001 v_vfsgen.. 00000000
v_lock.... 00000000 v_lock@... 09D0B574 v_vfsp.... 056D183C
v_mvfsp... 056D18D8 v_gnode... 09D0B5A8 v_next.... 00000000
v_vfsnext. 09D0A230 v_vfsprev. 09D0C0F0 v_pfsvnode 00000000
v_audit... 00000000

VFS_VNODES LIST...
              COUNT VFSGEN    GNODE     VFSP  DATAPTR TYPE FLAGS

   1 09D95500     0      0 09D95540 056D18D8 00000000 REG
   2 09D94AC0     0      0 09D94B00 056D18D8 00000000 DIR
   3 09D91DE8     0      0 09D91E28 056D18D8 00000000 REG
   4 09D91A10     0      0 09D91A50 056D18D8 00000000 DIR
   5 09D8EFC8     0      0 09D8F008 056D18D8 00000000 REG
   6 09D8EBF0     0      0 09D8EC30 056D18D8 00000000 DIR
   7 09D8C580     0      0 09D8C5C0 056D18D8 00000000 REG
   8 09D8C060     0      0 09D8C0A0 056D18D8 00000000 DIR
   9 09D8A058     0      0 09D8A098 056D18D8 00000000 REG
  10 09D89C80     0      0 09D89CC0 056D18D8 00000000 DIR
  11 09D89240     0      0 09D89280 056D18D8 00000000 REG
...
             COUNT VFSGEN    GNODE     VFSP  DATAPTR TYPE FLAGS

  63 09D73478     0      0 09D734B8 056D18D8 00000000 REG
  64 09D730A0     0      0 09D730E0 056D18D8 00000000 DIR
  65 09D0BFA8     1      0 09D0BFE8 056D18D8 00000000 DIR  ROOT
```

## specnode subcommand

### Purpose

The **specnode** subcommand displays the special device node structure at the specified address.

### Syntax

**specnode** *address*

### Parameters

- *address* – Specifies the effective address of a special device node structure. Use symbols, hexadecimal values, or hexadecimal expressions to specify the address.

### Aliases

**specno**

### Example

The following is an example of how to use the **specno** alias for the **specnode** subcommand:

```
KDB(0)> file 108  //print file entry
     ADDR     COUNT           OFFSET     DATA TYPE   FLAGS

 108 10001410     1 0000000000000000 32ABD1DC VNODE  WRITE NOCTTY

f_flag......... 00000802 f_count........ 00000001
f_options.......... 0000 f_type............ 0001
f_data......... 32ABD1DC f_offset... 0000000000000000
f_dir_off...... 00000000 f_cred......... 32BB5600
f_lock@........ 10001430 f_lock......... 00000000
f_offset_lock@. 10001434 f_offset_lock.. 00000000
f_vinfo........ 00000000 f_ops.......... 006A2F98 vnodefops+000000
VNODE.......... 32ABD1DC
v_flag.... 00000000 v_count... 00000018 v_vfsgen.. 00000000
v_lock.... 00000000 v_lock@... 32ABD1E8 v_vfsp.... 01FB4000
v_mvfsp... 00000000 v_gnode... 32843080 v_next.... 00000000
v_vfsnext. 00000000 v_vfsprev. 00000000 v_pfsvnode 14546080
v_audit... 00000000
KDB(0)> gno 32843080  //print gnode node entry
GNODE........... 32843080 32843080
gn_type....... 00000009 gn_flags...... 00000000 gn_seg........ 007FFFFF
gn_mwrcnt..... 00000000 gn_mrdcnt..... 00000000 gn_rdcnt...... 00000000
gn_wrcnt...... 00000000 gn_excnt...... 00000000 gn_rshcnt..... 00000000
gn_vnode...... 32ABD1DC gn_rdev....... 00040000 gn_ops........ spec_vnops
gn_chan....... 00000000 gn_reclk_lock. 00000000 gn_reclk_lock@ 328430B4
gn_reclk_event FFFFFFFF gn_filocks.... 00000000 gn_data....... 32843070
gn_type....... MPC
KDB(0)> specno 32843070  //print special node entry
SPECNODE........ 32843070
sn_next...... 00000000 sn_gen....... 00000537 sn_count..... 0001
sn_gnode.... @32843080 sn_pfsgnode.. 145460C0 sn_lock..... @3284307C 00000000
sn_attr...... 328560C0 sn_dev....... 00040000 sn_chan...... 00000000
sn_vnode..... 32ABD1DC sn_ops....... 006D9990 sn_type...... 00000009
sn_data...... 328439A8 fdev_chain_f. 00000000 sn_type...... MPC
sn_mode...... 00002192 sn_uid....... 00000000 sn_gid....... 00000000
sn_atime..... 4002A299 sec 02AB0F09 nsec
sn_mtime..... 40402524 sec 2C8B386B nsec
sn_ctime..... 40402524 sec 2C8B386B nsec
sn_acl....... 00000000

SN_VNODE........ 32ABD1DC
v_flag.... 00000000 v_count... 00000018 v_vfsgen.. 00000000
v_lock.... 00000000 v_lock@... 32ABD1E8 v_vfsp.... 01FB4000
```

```
v_mvfsp... 00000000 v_gnode... 32843080 v_next.... 00000000
v_vfsnext. 00000000 v_vfsprev. 00000000 v_pfsvnode 14546080
v_audit... 00000000

SN_GNODE......... 32843080
gn_type....... 00000009 gn_flags...... 00000000 gn_seg........ 007FFFFF
gn_mwrcnt..... 00000000 gn_mrdcnt..... 00000000 gn_rdcnt...... 00000000
(0)> more (^C to quit) ?
gn_wrcnt...... 00000000 gn_excnt...... 00000000 gn_rshcnt..... 00000000
gn_vnode...... 32ABD1DC gn_rdev....... 00040000 gn_ops........ spec_vnops
gn_chan....... 00000000 gn_reclk_lock. 00000000 gn_reclk_lock@ 328430B4
gn_reclk_event FFFFFFFF gn_filocks.... 00000000 gn_data....... 32843070
gn_type....... MPC

SN_PFSGNODE...... 145460C0
gn_type....... 00000004 gn_flags...... 00000000 gn_seg........ 00000000
gn_mwrcnt..... 00000000 gn_mrdcnt..... 00000000 gn_rdcnt...... 00000000
gn_wrcnt...... 00000000 gn_excnt...... 00000000 gn_rshcnt..... 00000000
gn_vnode...... 14546080 gn_rdev....... 00040000 gn_ops........ jfs_vops
gn_chan....... 00000000 gn_reclk_lock. 00000000 gn_reclk_lock@ 145460F4
gn_reclk_event FFFFFFFF gn_filocks.... 00000000 gn_data....... 145460B0
gn_type....... CHR
KDB(0)>
```

# devnode subcommand

## Purpose

The **devnode** subcommand displays device node table entries.

## Syntax

**devnode** [*slot* | *effectiveaddress*]

## Parameters

- *slot* – Specifies the slot number of a device node table entry. This parameter must be a decimal value.
- *effectiveaddress* – Specifies the effective address of a device node table entry. Use symbols, hexadecimal values, or hexadecimal expressions to specify the address.

If no parameter is entered, a summary is displayed with one line per table entry. Display detailed information for individual devnode table entries by specifying either a slot number or an address.

## Aliases

**devno**

## Example

The following is an example of how to use the **devnode** subcommand:

```
(0)> devnode  //print device node table
                   DEV CNT SPECNODE    GNODE    LASTR    PDATA TYPE

   1 0B241758 00300000   1 0B2212E0 0B241768 00000000 05CB4E00 CHR
   2 0B221C18 00100000   1 00000000 0B221C28 00000000 00000000 CHR
   3 0B221940 00110000   2 00000000 0B221950 00000000 00000000 BLK
   4 0B221870 00020000   1 0B221140 0B221880 00000000 00000000 CHR
   5 0B7E5A10 00120001   2 00000000 0B7E5A20 00000000 00000000 BLK
   6 0B241070 00020001   1 0B8A3EF0 0B241080 00000000 00000000 CHR
   7 0B2219A8 00020002   1 0B221008 0B2219B8 00000000 00000000 CHR
   8 0B2218D8 00130000   1 00000000 0B2218E8 00000000 00000000 CHR
   9 0B7E5BB0 00330001   1 00000000 0B7E5BC0 00000000 00000000 BLK
  10 0B221A10 00130001   1 00000000 0B221A20 00000000 00000000 CHR
  11 0B241008 00330002   1 00000000 0B241018 00000000 00000000 BLK
  12 0B7E59A8 00130002   1 00000000 0B7E59B8 00000000 00000000 CHR
  13 0B7E5C18 00330003   1 00000000 0B7E5C28 00000000 00000000 BLK
  14 0B7E5808 00130003   1 00000000 0B7E5818 00000000 00000000 CHR
  15 0B7E5A78 00330004   1 00000000 0B7E5A88 00000000 00000000 BLK
  16 0B7E5C80 00330005   1 00000000 0B7E5C90 00000000 00000000 BLK
  17 0B7E5CE8 00330006   1 00000000 0B7E5CF8 00000000 00000000 BLK
  18 0B2416F0 00040000   1 0B2211A8 0B241700 00000000 00000000 MPC
  19 0B221BB0 00150000   3 0B221688 0B221BC0 00000000 05CC3E00 CHR
  20 0B2410D8 00060000   1 0B221480 0B2410E8 00000000 00000000 CHR
(0)> more (^C to quit) ? //Interrupted
(0)> devno 3  //print device node slot 3
                   DEV CNT SPECNODE    GNODE    LASTR    PDATA TYPE

   3 0B221940 00110000   2 00000000 0B221950 00000000 00000000 BLK

forw...... 00DD6CD8 back...... 00DD6CD8 lock...... 00000000

GNODE........... 0B221950
gn_type....... 00000003 gn_flags...... 00000000 gn_seg........ 007FFFFF
gn_mwrcnt..... 00000000 gn_mrdcnt..... 00000000 gn_rdcnt...... 00000000
gn_wrcnt...... 00000002 gn_excnt...... 00000000 gn_rshcnt..... 00000000
gn_vnode...... 00000000 gn_rdev....... 00110000 gn_ops........ 00000000
gn_chan....... 00000000 gn_reclk_lock. 00000000 gn_reclk_lock@ 0B221984
```

```
gn_reclk_event 00000000 gn_filocks.... 00000000 gn_data....... 0B221940
gn_type....... BLK

SPECNODES....... 00000000
```

## fifonode subcommand

### Purpose

The **fifonode** subcommand displays fifo node table entries

### Syntax

**fifonode** [*slot* | *effectiveaddress*]

### Parameters

- *slot* – Specifies the slot number of a fifo node table entry. This parameter must be a decimal value.
- *effectiveaddress* – Specifies the effective address of a fifo node table entry. Use symbols, hexadecimal values, or hexadecimal expressions to specify the address.

If no parameter is entered, a summary with one line per entry is displayed. Display detailed information for individual entries with either a slot number or an address.

### Aliases

**fifono**

### Example

The following is an example of how to use the **fifono** alias for the **fifonode** subcommand:

```
(0)> fifono  //print fifo node table
              PFSGNODE SPECNODE     SIZE  RCNT  WCNT TYPE FLAG

   1 056D1C08 09D15EC8 0B2210D8 00000000     1     1 FIFO WWRT
   2 056D1CA8 09D1BB08 0B7E5070 00000000     1     1 FIFO RBLK WWRT
(0)> fifono 1 //print fifo node slot 1
              PFSGNODE SPECNODE     SIZE  RCNT  WCNT TYPE FLAG

   1 056D1C08 09D15EC8 0B2210D8 00000000     1     1 FIFO WWRT

ff_forw.... 00DD6D44 ff_back.... 00DD6D44 ff_dev..... FFFFFFFF
ff_poll.... 00000001 ff_rptr.... 00000000 ff_wptr.... 00000000
ff_revent.. FFFFFFFF ff_wevent.. FFFFFFFF ff_buf..... 056D1C34

SPECNODE........ 0B2210D8
sn_next...... 00000000 sn_count..... 00000001 sn_lock...... 00000000
sn_gnode..... 0B2210E8 sn_pfsgnode.. 09D15EC8 sn_attr...... 00000000
sn_dev....... FFFFFFFF sn_chan...... 00000000 sn_vnode..... 056CE070
sn_ops....... 002751B0 sn_devnode... 056D1C08 sn_type...... FIFO

SN_VNODE........ 056CE070
v_flag.... 00000000 v_count... 00000002 v_vfsgen.. 00000000
v_lock.... 00000000 v_lock@... 056CE07C v_vfsp.... 01AC9810
v_mvfsp... 00000000 v_gnode... 0B2210E8 v_next.... 00000000
v_vfsnext. 00000000 v_vfsprev. 00000000 v_pfsvnode 09D15E88
v_audit... 00000000

SN_GNODE......... 0B2210E8
gn_type....... 00000008 gn_flags...... 00000000 gn_seg........ 007FFFFF
gn_mwrcnt..... 00000000 gn_mrdcnt..... 00000000 gn_rdcnt...... 00000000
gn_wrcnt...... 00000000 gn_excnt...... 00000000 gn_rshcnt..... 00000000
gn_vnode...... 056CE070 gn_rdev....... FFFFFFFF gn_ops........ fifo_vnops
gn_chan....... 00000000 gn_reclk_lock. 00000000 gn_reclk_lock@ 0B22111C
gn_reclk_event 00000000 gn_filocks.... 00000000 gn_data....... 0B2210D8
gn_type....... FIFO

SN_PFSGNODE...... 09D15EC8
```

```
gn_type....... 00000008 gn_flags...... 00000000 gn_seg........ 00000000
gn_mwrcnt..... 00000000 gn_mrdcnt..... 00000000 gn_rdcnt...... 00000000
gn_wrcnt...... 00000000 gn_excnt...... 00000000 gn_rshcnt..... 00000000
gn_vnode...... 09D15E88 gn_rdev....... 000A0005 gn_ops........ jfs_vops
gn_chan....... 00000000 gn_reclk_lock. 00000000 gn_reclk_lock@ 09D15EFC
gn_reclk_event FFFFFFFF gn_filocks.... 00000000 gn_data....... 09D15EB8
gn_type....... FIFO
```

# hnode subcommand

## Purpose

The **hnode** subcommand displays hash node table entries.

## Syntax

**hnode** [*bucket* | *effectiveaddress*]

## Parameters

- *bucket* – Specifies the bucket number within the hash node table. This parameter must be a decimal value.
- *effectiveaddress* – Specifies the effective address of a bucket in the hash node table. Use symbols, hexadecimal values, or hexadecimal expressions to specify the address.

If no parameter is entered, a summary that contains one line per hash bucket is displayed. Display the entries for a specific bucket by specifying the bucket number or the address of the bucket.

## Aliases

**hno**

## Example

The following is an example of how to use the **hno** alias for the **hnode** subcommand:

```
(0)> hno  //print hash node table
                  BUCKET HEAD     LOCK     COUNT

hnodetable+000000    1   0B241758 00000000     2
hnodetable+0000C0   17   0B221940 00000000     1
hnodetable+00012C   26   056D1C08 00000000     1
hnodetable+000180   33   0B221870 00000000     1
hnodetable+00018C   34   0B7E5A10 00000000     2
hnodetable+000198   35   0B2219A8 00000000     1
hnodetable+000240   49   0B2218D8 00000000     1
hnodetable+00024C   50   0B7E5BB0 00000000     2
hnodetable+000258   51   0B241008 00000000     2
hnodetable+000264   52   0B7E5C18 00000000     2
hnodetable+000270   53   0B7E5A78 00000000     1
hnodetable+00027C   54   0B7E5C80 00000000     1
hnodetable+000288   55   0B7E5CE8 00000000     1
hnodetable+000300   65   0B2416F0 00000000     1
hnodetable+0003C0   81   0B221BB0 00000000     1
hnodetable+000480   97   0B2410D8 00000000     1
hnodetable+00048C   98   0B221B48 00000000     1
hnodetable+000540  113   0B7E5AE0 00000000     1
hnodetable+00054C  114   0B7E5EF0 00000000     1
hnodetable+000600  129   0B7E5B48 00000000     1
(0)> more (^C to quit) ?  //Interrupted
(0)> hno 34  //print hash node bucket 34
HASH ENTRY( 34): 00DD6DA4
                   DEV CNT SPECNODE    GNODE    LASTR    PDATA TYPE

   1 0B7E5A10 00120001   2 00000000 0B7E5A20 00000000 00000000 BLK
   2 0B241070 00020001   1 0B8A3EF0 0B241080 00000000 00000000 CHR
```

# jfsnode subcommand

## Purpose

The **jfsnode** subcommand prints details of the inode pool when no input parameter is provided. If the address of a jfs node is provided as an input parameter, the **jfsnode** subcommand verifies the jfs node and gives additional information on the related file system.

**Note:** This subcommand is only available in the **kdb** command.

## Syntax

**jfsnode** [*address*]

## Parameters

- *address* – Specifies the address of a node allocated in the inode cache.

**Note:** The *address* parameter is useful only for nodes allocated in the inode cache. It is not useful for soft mounts, specnodes, cdrnodes, or other non-jfs structures.

## Aliases

**jno**

## Example

The following is an example of how to use the **jfsnode** subcommand:

```
0)> jfsnode
INODES pool starts at 0x1101D6058
Static table[0] starts at 0xF100009E14793000, ends at 0xF100009E149C3000
Static table[1] starts at 0xF100009E149C3000, ends at 0xF100009E14BF3000
Static table[2] starts at 0xF100009E14BF3000, ends at 0xF100009E14E23000
Static table[3] starts at 0xF100009E14E23000, ends at 0xF100009E15053000
Static table[4] starts at 0xF100009E15053000, ends at 0xF100009E15283000
Static table[5] starts at 0xF100009E15283000, ends at 0xF100009E154B3000
Static table[6] starts at 0xF100009E154B3000, ends at 0xF100009E156E3000
Static table[7] starts at 0xF100009E156E3000, ends at 0xF100009E15913000
Static table[8] starts at 0xF100009E15913000, ends at 0xF100009E15B43000
Static table[9] starts at 0xF100009E15B43000, ends at 0xF100009E15D73000
Static table[10] starts at 0xF100009E15D73000, ends at 0xF100009E15FA3000
Static table[11] starts at 0xF100009E15FA3000, ends at 0xF100009E161D3000
Static table[12] starts at 0xF100009E161D3000, ends at 0xF100009E16403000
Static table[13] starts at 0xF100009E16403000, ends at 0xF100009E16633000
Static table[14] starts at 0xF100009E16633000, ends at 0xF100009E16863000
Static table[15] starts at 0xF100009E16863000, ends at 0xF100009E16A93000
Static table[16] starts at 0xF100009E16A93000, ends at 0xF100009E16CC3000
Static table[17] starts at 0xF100009E16CC3000, ends at 0xF100009E16EF3000
Static table[18] starts at 0xF100009E16EF3000, ends at 0xF100009E17123000
Static table[19] starts at 0xF100009E17123000, ends at 0xF100009E17353000
Static table[20] starts at 0xF100009E17353000, ends at 0xF100009E17583000
Static table[21] starts at 0xF100009E17583000, ends at 0xF100009E177B3000
Static table[22] starts at 0xF100009E177B3000, ends at 0xF100009E179E3000
Static table[23] starts at 0xF100009E179E3000, ends at 0xF100009E17C13000
Static table[24] starts at 0xF100009E17C13000, ends at 0xF100009E17E43000
Static table[25] starts at 0xF100009E17E43000, ends at 0xF100009E18073000
Static table[26] starts at 0xF100009E18073000, ends at 0xF100009E182A3000
Static table[27] starts at 0xF100009E182A3000, ends at 0xF100009E184D3000
Static table[28] starts at 0xF100009E184D3000, ends at 0xF100009E18703000
Static table[29] starts at 0xF100009E18703000, ends at 0xF100009E18933000
Static table[30] starts at 0xF100009E18933000, ends at 0xF100009E18B63000
Static table[31] starts at 0xF100009E18B63000, ends at 0xF100009E18D93000
Object (xnode) size is 0x230
```

```
vnode offset is 0x0
inode offset is 0x58
gnode offset is 0x78
(0)>
(0)> jno 0xF100009E18B63000
0xF100009E18B63000 is a vnode
           INODE            GNODE            VNODE              VFS  FILESYSTEM
F100009E18B63058 F100009E18B63078 F100009E18B63000                0
(0)> jno 0xF100009E18B63005
0xF100009E18B63005 is an OFFSET into the vnode at 0xF100009E18B63000
           INODE            GNODE            VNODE              VFS  FILESYSTEM
F100009E18B63058 F100009E18B63078 F100009E18B63000                0
(0)> jno 0x1
Address not in jfs inode cache: 0x1
(0)>
```

## kfset subcommand

### Purpose

The **kfset** subcommand displays the **kdm fset cache data** structure.

### Syntax

**kfset** *address*

### Parameters

- *address* – Identifies the address of the **kdm fset cache data** structure to display.

### Aliases

**kfs**

### Example

The following is an example of how to use the **kfset** subcommand:

```
KDB(0)> kfset 0x328A5400
linknxt.0x01FEB540  linkprv.0x01FEB540
fsid....0x00000000002C0007             refcnt..0x00000000  enables.0x80000028
evdisp@.0x2FFBB394
kvnode..0x3173F1E0  fsetops.0x007FE300  attrnxt.0x328A5598  attrprv.0x328A5598
lock@...0x2FFBB520  options.0x00000000
mpath...0x3006A060  mplen...5           dpath...0x3006A0B0  dplen...12
attrnam.[       ]  class...0x00000000  subcls..0x00000000  length..0

Note: The kfset pointer is in the kdm vnode structure and may be
obtained from the output of the kvnode command, in the fset field:

KDB(0)> kvnode 0x3173F1E0
enables..0x00000000  flags....0x00000000  nreg.....0x00000000
op.......0x007FE320  fset.....0x328A5400
regp.....0x00000000  data.....0x328389D8
```

# Chapter 31. Display Enhanced Journaled File System information subcommands

The subcommands in this category can be used to display Enhanced Journaled File System (JFS2) information. These subcommands include the following:

- j2
- i2
- tree
- dt
- xt
- pgbuf
- pgobj
- txblock
- txblocki
- txlock
- j2trace
- bmblock
- j2no
- j2logbuf
- j2logx
- j2log
- pile
- slab

# j2 subcommand

## Purpose

The **j2** subcommand temporarily sets up access to the Enhanced Journaled File System (JFS2) metadata buffers so that the command specified as an input parameter can run correctly.

## Syntax

**j2** *cmd*

## Parameters

- *cmd* – Indicates the actual **kdb** or KDB kernel debugger **j2** subcommand that you want to run.

On the 32-bit kernel, there is a separate segment for JFS2 metadata. The **j2** subcommand sets up the segment registers so that any command dealing with JFS2 metadata can examine the address in question as if JFS2 were attached.

On the 64-bit kernel, no address space setup is necessary, so the **j2** subcommand runs the specified command.

The **j2** subcommand is a wrapper that establishes the proper run environment for the specified subcommand that requires access to the **j2** metadata.

## Aliases

**jfs2**

## Example

The following is an example of how to use the **j2** subcommand:

```
j2 dd 0xD0000000 20 //displays 32 words from the first page of the metadata segment
KDB(0)> j2 dd 0xD0000000 20
D0000000: 4845415000043000 0000000000000001  HEAP..0.........
D0000010: 0000FFBD00000000 0000000000000000  ................
D0000020: 0000000000000000 0000000000000000  ................
D0000030: 0000000000000000 0000000000000000  ................
D0000040: 0000000000000000 0000000000000000  ................
D0000050: 0000000000000000 0000000000000000  ................
D0000060: 0000000000000000 0000000000000000  ................
D0000070: 0000000000000000 0000000000000000  ................
D0000080: 0000000000000000 00FFFFFF00FFFFFF  ................
D0000090: 00FFFFFF00FFFFFF 00FFFFFF00FFFFFF  ................
D00000A0: 00FFFFFF00FFFFFF 00FFFFFF00FFFFFF  ................
D00000B0: 00FFFFFF00FFFFFF 4845415000042F48  ........HEAP../H
D00000C0: 0000000000000000 0000FFBD0000D000  ................
D00000D0: 0000000000000000 0000000000000000  ................
D00000E0: 0000000000000000 0000000000000000  ................
D00000F0: 0000000000000000 0000000000000000  ................
KDB(0)>
```

# i2 subcommand

## Purpose

The **i2** subcommand displays the Enhanced Journaled File System (JFS2) inode.

## Syntax

**i2** [*address* | **-c**]

**i2** [**-d** *device*] [**-i** *inumber*] [**-m** *count*]

## Parameters

- *address* – Displays the JFS2 inode structure at the specified inode address.
- **-c** – Displays the inode cache table.
- **-d** *device* – Displays a list of inodes in the specified device.
- **-i** *inumber* – Displays the inode structure of the inode number specified.
- **-m** *count* – Displays a list of inodes with a minimum number of the open count specified.

The **-d**, **-i**, and **-m** flags can be mixed. For these three flags, when multiple inodes satisfy the criteria, only summary information is displayed. If a single inode satisfies the criteria, detailed information is also displayed.

When the **i2** command is invoked without any parameters, a summary list of inodes in memory is displayed along with the inodes' address, device, and inode number.

## Aliases

**inode2**

## Example

The following is an example of how to use the **i2** subcommand:

```
KDB(0)> i2
ADDRESS     DEVICE    I_NUM      IPMNT          COUNT   TYPE    FLAG
325A8080   000A000B     2       3252F080        00001   VDIR
32573080   000A000B     2       3252F080        00001   NON
32584080   000A000A     0       00000000        00001   NON
32563080   000A000B     1       3252F080        00001   NON
3252F080   000A000B     mounted 3252F080        00001   NON
32595400   000A000B     6       3252F080        00000   VDIR    CNEW
325D1080   000A000B     5       3252F080        00000   VREG    UPDNEW
325C1080   000A000B     4       3252F080        00000   VREG    UPDNEW
32595080   000A000B     16      3252F080        00001   NON     CDIRTY
32584400   000A000B     35      3252F080        00000   VREG    UPDNEW
32573400   000A000B     34      3252F080        00000   VREG    UPDNEW
32563400   000A000B     33      3252F080        00000   VREG    UPDNEW
325E1080   000A000B     32      3252F080        00001   VDIR
3252F400   000A000B     64      3252F080        00000   VDIR    CNEW
KDB(0)>i2 325C1080
ADDRESS     DEVICE    I_NUM      IPMNT          COUNT   TYPE    FLAG
325C1080   000A000B     4       3252F080        00000   VREG    UPDNEW

In-memory Working Inode:
hashClass....0x000002AF  cacheClass...0x00000007  count........0x00000000
capability...0x000001B7  atlhead......0x00000000  atltail......0x00000000
bxflag.......0x00000000  blid.........0x00000000  btindex......0x00000002
diocnt.......0x00000000  nondiocnt....0x00000000
dev..........0x000A000B  synctime.....0x403CE9A8  nodelock.....0x00000000
ipmnt........0x3252F080  ipimap.......0x32595080  pagerObject..0x31A6D000
```

```
event........0xFFFFFFFF  fsevent......0xFFFFFFFF  openevent....0xFFFFFFFF
cacheLst.nxt.0x317230B0  cacheLst.prv.0x317230B0  freeNext.....0x317230B0
hashLst.nxt..0x00000000  hashLst.prv..0x31BA1034  kdmvp........0x00000000
flag.........0x00000000
cflag........0x00000000
xlock........0x00000000
fsxlock......0x00000000
btorder......0x00000000
agstart......0x0000000000000000
lastCommittedSize...0x0000000000001000

Pseudo pagerBuffer @ 0x325C1124:
(0)> more (^C to quit) ?
```

## tree subcommand

### Purpose

The **tree** subcommand displays either the Enhanced Journaled File System (JFS2) d-tree or x-tree structure based on the specified inode parameter.

### Syntax

**tree** *address*

### Parameters

- *address* – Specifies the address of an inode. If the address of the specified inode is a directory, the d-tree structure is displayed. If the address of the specified inode is not a directory, the x-tree structure is displayed. This is a required parameter.

### Aliases

No aliases.

### Example

The following is an example of how to use the **tree** subcommand:

```
KDB(0)> tree 325C1080
flag.........0x83
flag_name....BT_ROOT  BT_LEAF
nextindex....3
maxentry.....18
self.len.....0
self.addr1...0x00
self.addr2...0x00000000
self.addr....0
next.........0x34E0
prev.........0x34E0

Leaf xads:
xad[2]
flag.........0x00
len..........1
addr1........0x00
addr2........0x00000028
off1.........0x00
off2.........0x00000000
offset.......0
address......40

xtree: Press [s]elect or e[x]it >
```

# dtree subcommand

## Purpose

The **dtree** subcommand displays the Enhanced Journaled File System (JFS2) d-tree structure and allows the user to walk the **dtree** structure.

## Syntax

**dtree** *address*

## Parameters

- *address* – Specifies the address of the d-tree structure.

The **dtree** subcommand contains its own subcommands that allow the user to walk the d-tree.

| Subcommand | Function |
| --- | --- |
| **f** | Walks **freelist** entries. |
| **s** | Displays the specified slot entry. |
| **t** | Displays the formatted **stbl** structure. |
| **u** | Visits the parent node (but not the parent directory). |
| **c** | Visits the child node. |
| **x** | Exits subcommand mode. |

## Aliases

**dt**

## Example

The following is an example of how to use the **dt** alias for the **dtree** subcommand:

```
KDB(0)> dt 0x325E1248
Internal D-tree page:
flag.........0x85
flag_name....BT_ROOT  BT_INTERNAL
freecnt......7
Actual Free Count: 7
nextindex....1
freelist.....2
self.len.....0x010203
maxslot......0
stblindex....0
self.addr1...0x04
self.addr2...0x05060708
next.........0x2
prev.........0x0

dtree: [n]ext, [f]reelist, [s]lot, s[t]bl, or e[x]it >
```

# xtree subcommand

## Purpose

The **xtree** subcommand displays the Enhanced Journaled File System (JFS2) **xtree** structure and allows the user to walk the **xtree** structure.

## Syntax

**xtree** *address*

## Parameters

- *address* – Displays the x-tree at the address of the specified x-tree.

The **dtree** subcommand contains its own subcommands that allow the user to walk the **x-tree** structure.

| Subcommand | Function |
| --- | --- |
| **s** | Selects the **xad** entry to view. |
| **u** | Visits the parent node. |
| **c** | Visits the child node. |
| **x** | Exits subcommand mode. |

## Aliases

**xt**

## Example

The following is an example of how to use the **xtree** subcommand:

```
KDB(0)> xtree 0x325C1248
flag.........0x83
flag_name....BT_ROOT  BT_LEAF
nextindex....3
maxentry.....18
self.len.....0
self.addr1...0x00
self.addr2...0x00000000
self.addr....0
next.........0x34E0
prev.........0x34E0

Leaf xads:
xad[2]
flag.........0x00
len..........1
addr1........0x00
addr2........0x00000028
off1.........0x00
off2.........0x00000000
offset.......0
address......40

xtree: Press [s]elect or e[x]it >
```

# pgobj subcommand

## Purpose

The **pgobj** subcommand displays the Enhanced Journaled File System (JFS2) pager object structure.

## Syntax

**pgobj** *address*

## Parameters

- *address* – Displays the pager object structure at the specified address.

## Aliases

No aliases.

## Example

The following is an example of how to use the **pgobj** subcommand:

```
KDB(0)> pgobj 0x325B9000
flags........0x00000000  mCount.......0x00000000  cacheClass...0xFFFFFFFF
fileObject...0x325E1080  pageList.....0x325405C4  freeNext.....0x31C8F000
pagerDevice..0x31C8F000  lock.........0x00000000  ioWait.......0xFFFFFFFF
deleteWait...0xFFFFFFFF  xWait........0xFFFFFFFF  mWaitShared..0xFFFFFFFF
mWaitExcl....0xFFFFFFFF  pLastRead....0x00000000FFFFFFFF  pTripWire....0x00000000FFFFFFFF
l2LastReadAhead...........0x00  l2LastLastReadAhead........0x00
po_randReadTrust.....0x00000000  nPageLock............0x00000000
cWriteBehind.0x0000000000000000  nRandomWrite.........0x00000000

RBNA:
rbnaXoffset..0x0000000000000000  rbnaXlen..0x00000000
rbnaDelta....              0x00  nRbnaXad..0xFFFFFFFF

wipXAD:
flag.........0x00000000
len..........0x00000000  addr1........0x00000000  addr2........0x00000000
off1.........0x00000000  off2.........0x00000000
offset.......0x0000000000000000  address......0x0000000000000000

[l]ist pagerBuffer page list, e[x]it >
```

## pgbuf subcommand

### Purpose

The **pgbuf** subcommand displays the Enhanced Journaled File System (JFS2) pager buffer structure.

### Syntax

**pgbuf** *address* | **-c**

### Parameters

- *address* – Displays the JFS2 pager buffer structure at the specified address.
- **-c** – Displays a list of the JFS2 pager buffers in the buffer cache.

### Aliases

No aliases.

### Example

The following is an example of how to use the **pgbuf** subcommand:

```
KDB(0)> pgbuf -c
jCache:
  nBuffer: 0xA00 (2560)
  nCacheClass:  9
  minFreePerCC: 5
  maxFreePerCC: 8
  nHashClass:   0x3FF (1023)
  cacheTable:   0x31A70000
  hashTable:    0x31C0E000
  freeWait:     0xFFFFFFFF
jCacheClassLow: 0

Cache table:
   CLASS         BUFS      FREE       LRU    CACHELIST.HEAD     FREELIST.HEAD
       0            3         1         0          3253F8DC          3253F090
       1            3         1         0          32540214          3253F17C
       2            3         0         0          3253F268                 0
       3            3         0         0          3253F354                 0
       4            3         0         0          3253F440                 0
       5            3         0         0          3253F52C                 0
       6            2         0         0          3253FE64                 0
       7            2         0         0          3253FF50                 0
       8            2         0         0          3253F7F0                 0
KDB(0)>pgbuf 3253F8DC
xflag........0x0000000C  BUFFER PAGE
nohomeok.....0x00000000
lid..........0x00000000
flags........0x00000011  METADATA IODONE
count................0x00000000  cacheClass...........0x00000000
data.........0xD004C000  syncList.nxt.0x00000000  syncList.prv.0x00000000
logx.........0x00000000  ip...........0x32595080  pagerObject..0x325A7000
pageList.nxt.0x00000000  pageList.prv.0x3253FA08  hashList.nxt.0x00000000
hashList.prv.0x31C10460  cacheLst.nxt.0x32540128  cacheLst.prv.0x31A70010
ioListNext...0x32540128  freeList.nxt.0x32540128  freeList.prv.0x31A70010
ioNext.......0x00000000  iobp.........0x3253F884  ioWait.......0xFFFFFFFF
waitList.....0xFFFFFFFF
lsn..........0x0000000000000000  clsn.........0x0000000000000000
xoffset......0x0000000000000000  pxd..........0x000000000000001E
KDB(0)>
```

# txblock subcommand

## Purpose

The **txblock** subcommand displays the Enhanced Journaled File System (JFS2) transaction block structure.

## Syntax

**txblock** *address*

## Parameters

- *address* – Displays the transaction block at the specified address.

## Aliases

**txblk**

## Example

The following is an example of how to use the **txblock** subcommand:

```
KDB(3)> txblock 32503108
xflag........0x00000000  flag.........0x00000000  next.........0x00000000
locker.......0x00000000  eor..........0x00000000  logTid.......0x00000005
lidList......0x2FF3ABA8  waitor.......0xFFFFFFFF  lwmbp........0x00000000
bp...........0x00000000  cqnext.......0x00000000  gcWait.......0xFFFFFFFF
ipmnt........0x325C1780         lwmlsn.......0x0000000000000000
clsn.........0x0000000000000000  lspn.........0x0000000000000000
KDB(3)>
```

## txblocki subcommand

### Purpose

The **txblocki** subcommand displays the Enhanced Journaled File System (JFS2) transaction block.

### Syntax

**txblocki** *index*

### Parameters

- *index* – Displays the transaction block at the specified index.

### Aliases

**txblki**

### Example

The following is an example of how to use the **txblocki** subcommand:

```
KDB(0)> txblocki 1
xflag........0x00000000  flag.........0x00000000  next.........0x00000000
locker.......0x00000000  eor..........0x00000000  logTid.......0x00000005
lidList......0x2FF3ABA8  waitor.......0xFFFFFFFF  lwmbp........0x325411C0
bp...........0x00000000  cqnext.......0x00000000  gcWait.......0xFFFFFFFF
ipmnt........0x325C1780          lwmlsn.......0x0000000000006F38
clsn.........0x0000000000000000  lspn.........0x0000000000000000
KDB(3)>
```

# txlock subcommand

## Purpose

The **txlock** subcommand displays the Enhanced Journaled File System (JFS2) transaction lock structure.

## Syntax

**txlock** *address*

## Parameters

- *address* – Displays the transaction lock structure at the specified address.

## Aliases

**txlck**

## Example

The following is an example of how to use the **txlock** subcommand:

```
KDB(3)> txlock 2FF3ABA8
tid..........0x00000003
flag.........0x00008801   PAGELOCK LOG LOCAL
next.........0x2FF3AB60  ip...........0x32573B00
bp...........0x325411C0  lock.........0x00000000
type.........0x00008002   GROW ENTRY INODE

maxcnt.......0x00000016  l2linesize...0x00000004  index........0x00000001

lv[0].offset.0x00000040  lv[0].length.0x00000008
next.........0x00000000
KDB(3)>
```

## j2trace subcommand

### Purpose

The **j2trace** subcommand displays the Enhanced Journaled File System (JFS2) trace table.

### Syntax

**j2trace**

### Parameters

No parameters.

### Aliases

**j2trc**, **j2t**

### Example

The following is an example of how to use the **j2trace** subcommand:

```
KDB(0)> j2trace
IDX  EVENT          X320     X640             X641             X642

0000 3010           00000000 0000000000002000 0000000000000000 0000000032584080
KDB(0)>
```

# bmblock subcommand

## Purpose

The **bmblock** subcommand displays the Enhanced Journaled File System (JFS2) metadata block and tries to lookup the hash value for a particular block and see if it exists in the cache.

## Syntax

**bmblock** *ipAddr xoff* block | page | raw

## Parameters

- *ipAddr* – Specifies the address of an inode.
- *xoff* – Specifies the offset.
- block, page, raw – Specifies the page buffer type. One of these values must be provided.

## Aliases

**bmb**, **bmblk**

## Example

The following is an example of how to use the **bmb** alias for the **bmblock** subcommand:

```
(0)> bmb 0xF10010F00F655C80 1C72F block
Hashclass @ F10010F00F4957D0
Pager buffer @ F10010F00F73C128
xflag........0x0000000A  BUFFER BLOCK
nohomeok.....0x00000000
lid..........0x0000000000000000
flags........0x00020011  METADATA IODONE HIT
count................0x00000000  cacheClass...........0x00000002
data.........0xF10010A11006E000  logx.........0x0000000000000000
syncList.nxt.0x0000000000000000  syncList.prv.0x0000000000000000
ip...........0xF10010F00F655C80  pagerObject..0xF10010F00F2BB0C8
pageList.nxt.0xF10010F00F310DE8  pageList.prv.0xF10010F00F73C330
hashList.nxt.0x0000000000000000  hashList.prv.0xF10010F00F4957D0
cacheLst.nxt.0xF10010F0107EE058  cacheLst.prv.0xF10010F00F8F22E8
freeList.nxt.0xF10010F0107EE058  freeList.prv.0xF10010F00F8F22E8
ioListNext...0xF10010F0107EE058  ioNext.......0x0000000000000000
iobp.........0xF10010F00F73C058  ioWait.......0xFFFFFFFFFFFFFFFF
waitList.....0xFFFFFFFFFFFFFFFF
lsn..........0x0000000000000000  clsn.........0x0000000000000000
xoffset......0x000000000001C72F  pxd..........0x000000000001C72F
```

## jfs2node subcommand

### Purpose

The **jfs2node** subcommand displays the Enhanced Journaled File System (JFS2) xnode structures.

### Syntax

**jfs2node** *address*

### Parameters

- *address* – Specifies an address at which to check whether that address is a valid JFS2 xnode structure or an offset into one. If there is a valid xnode or offset, the **jfs2node** subcommand displays the relevant structure. This is a required parameter.

### Aliases

**j2no**

### Example

The following is an example of how to use the **j2no** alias for the **jfs2node** subcommand:

```
(0)> j2no 0x1
0x1 is not a valid JFS2 xnode address.

(0)> i2
ADDRESS     DEVICE    I_NUM      IPMNT          COUNT   TYPE    FLAG
369F9080   00220001     1       369C9080        00001   NON
369C9080   00220001     mounted 369C9080        00001   NON
36A1F080   00220002     0       00000000        00001   NON
36A43080   00220001     2       369C9080        00001   VDIR
36A0C080   00220001     2       369C9080        00001   NON
36A30080   00220001     16      369C9080        00001   NON
(0)> j2no 36A1F080
0x36A1F080 is an inode:


In-memory Working Inode:
hashClass....0x00000422  cacheClass...0x00000005  count........0x00000001
capability...0x00000125  atlhead......0x00000000  atltail......0x00000000
bxflag.......0x00000000  blid.........0x00000000  btindex......0x00000000
diocnt.......0x00000000  nondiocnt....0x00000000
dev..........0x00220002  synctime.....0x00000000  nodelock.....0x00000000
ipmnt........0x00000000  ipimap.......0x00000000  pagerObject..0x00000000
event........0xFFFFFFFF  fsevent......0xFFFFFFFF  openevent....0xFFFFFFFF
cacheLst.nxt.0x00000000  cacheLst.prv.0x00000000  freeNext.....0x00000000
hashLst.nxt..0x00000000  hashLst.prv..0x366BE198  kdmvp........0x00000000
flag.........0x00001000  flag_type....SYSTEM
cflag........0x00000000
xlock........0x00000000
fsxlock......0x00000000
btorder......0x00000000
agstart......0x0000000000000000
lastCommittedSize...0x0000000000000000
.
.
.


(0)> j2no 0x36A1F085
0x36A1F085 is at offset 5 into wInode:
```

```
In-memory Working Inode:
hashClass....0x00000422  cacheClass...0x00000005  count........0x00000001
capability...0x00000125  atlhead......0x00000000  atltail......0x00000000
bxflag.......0x00000000  blid.........0x00000000  btindex......0x00000000
diocnt.......0x00000000  nondiocnt....0x00000000
.
.<as above>
.
```

## j2logbuf subcommand

### Purpose

The **j2logbuf** displays the Enhanced Journaled File System (JFS2) log buffer structure.

### Syntax

**j2logbuf** *address*

### Parameters

- *address* – Displays the JFS2 log buffer at the specified address.

### Aliases

No aliases.

### Example

The following is an example of how to use the **j2logbuf** subcommand:

```
KDB(0)> j2logbuf 31D6F5C4
lb_flags........0x0000210C  LB_WRITE LB_GC LB_IODONE LB_IOERROR
lb_lspn.........0x0000000000001246  lb_clsn.........0x0000000000000000
lb_ceor.........0x00000378  lb_blkno........0x0000000000000000
lb_pn...........0x00001246  lb_eor..........0x00000378
lb_log..........0x32557400  logx............0x31D6C000
syncList.nxt....0x00000000  syncList.prv....0x00000000
pageList.nxt....0x00000000  pageList.prv....0x00000000
hashList.nxt....0x00000000  hashList.prv....0x00000000
cacheLst.nxt....0x00000000  cacheLst.prv....0x00000000
freeList........0x00000000  ioNext..........0x31D6F5C4
waitList........0xFFFFFFFF  data............0xD005A000
ioWait..........0xFFFFFFFF  iobp............0x31D6F56C
KDB(0)>
```

## j2logx subcommand

### Purpose

The **j2logx** subcommand displays the **logx** structure.

### Syntax

**j2logx** [*address*]

### Parameters

- *address* – Displays the **logx** structure at the specified address.

### Aliases

No aliases.

### Example

The following is an example of how to use the **j2logx** subcommand:

```
KDB(0)> j2logx 31D6C000
flag.........0x00000000  count........0x00001E02  errCount.....0x00000204
hwmErrCount..0x00001000  lwmErrCount..0x00000020
lsn............0x0000000001246378  clsn............0x00000000000015BE
size...........0x0000000002000000  space...........0x01FFE000
syncpt.........0x00000000010C23B0 sync............0x00000000010C23B0
nFreeBuffer.....0x00000002  nBuffer.........0x00000001
hwmBuffer.......0x00000280  lwmBuffer.......0x00000140
pageOutQueue....0x31D6F5C4  freeBufferList..0x31C03440
lwmBufferWait...0xFFFFFFFF  freeBufferWait..0xFFFFFFFF
syncListLock....0x00000000  ioLock..........0x00000000
syncList.head...0x3283A9B8  syncList.tail...0x32833B4C
freeList........0x00000000  iLogSyncCursor.@.0x005A7198
bmLogSyncCursor.@.0x005A71C0 bmLogSyncRCursor.@.0x005A71E8
KDB(0)>
```

## j2log subcommand

### Purpose

The **j2log** subcommand displays the **log-t** structure.

### Syntax

**j2log** *address*

### Parameters

- *address* – Specifies a valid address for the **log-t** structure.

### Aliases

No aliases.

### Example

The following is an example of how to use the **j2log** subcommand:

```
KDB(0)> j2log 32557400
di_number....0x0000000000000000
di_gen.........0x00000000  di_fileset.....0x00000000
serial.......0x000000000000002C  base.........0x0000000000000000
flag.........0x00000100
state........0x00000004 LOGIOERROR
size.........0x00002000  bsize........0x00000000
pbsize.......0x00000000  l2bsize......0x0000  l2pbsize.....0x0009
logTid.......0x000007D0  lspn.........0x0000000000001246
pn...........0x00001246  eor..........0x00000378  cflag........0x00000000
gcrtc........0x00000000  syncState....0x00000000  nextsync.....0x001FFFF0
active.......0x00000000  syncBarrier..0x00000000  syncTid......0x00000004
after wInode, start at 0x005A70F8
bp...........0x31D6F5C4  dev..........0x000A000A
devfp........0x100038A0  logx.........0x31D6C000
logList.nxt..0x00000000  logList.prv..0x00806F7C
rdwrLock.....0x00000000  logLock......0x00000000
CMQ.head.....0x00000000  CMQ.tail.....0x00000000
gclrt........0x324F30B0  gcLock.......0x00000000
syncWait.....0x00000000  nTxLog.......0x00000000
KDB(0)>
```

# pile subcommand

## Purpose

The **pile** subcommand displays information about pile data structures.

## Syntax

**pile** [*address*]

## Parameters

- *address* – Specifies the memory address of the **pile** structure.

The **pile** subcommand can be run in the following ways:

- If no argument is specified, the **pile** subcommand lists the addresses of all the piles on the system and validates the pile identifier of the specified pile.
- If an address is specified, the **pile** subcommand attempts to print the contents of that address as a pile structure and validates the pile ID of every pile in the system.

If a valid pile identifier is not detected, an error message is displayed.

## Aliases

No aliases.

## Example

The following is an example of how to use the **pile** subcommand:

```
KDB(0)> pile
ADDRESS        NAME             cur_total_pages
0x3004B380     NLC64            0x0000000000000004
0x3004B400     NLC128           0x0000000000000000
0x3004B480     NLC256           0x0000000000000000
0x32BAE600     iCache           0x0000000000000010
0x32BAE580     iCache           0x0000000000000010
0x32BAE480     iCache           0x0000000000000010
0x32BAE500     iCache           0x0000000000000010
0x32BAE300     iCache           0x0000000000000010
0x32BAE380     bmIOBufPile      0x0000000000000000
0x32BAE680     bmXBufPile       0x0000000000000004
0x32BAE700     j2VCBufferPool   0x0000000000000000
0x32BAE780     j2VCBufferPool   0x0000000000000000
0x32BAE800     j2VCBufferPool   0x0000000000000000
0x32BAE880     j2VCBufferPool   0x0000000000000028
0x32BAE900     j2VCBufferPool   0x0000000000000000
0x32BAE980     dioCache         0x0000000000000004
0x32BAEA00     dioReq           0x0000000000000000
0x32BAEA80     dioPIOVPile      0x0000000000000000
KDB(0)> pile 0x3004B380

name........NLC64
prev........0x32BAEA80 next........0x3004B400
ID..........0x50494C45 objectsize..0x0044     align.......0x0003
slabsize....0x0004     intpri......0x000B     flags.......0x00000000
maxtotalpg..0xFFFFFFFFFFFFFFFF                mintotalpg..0x0000000000000000
curtotalpg..0x0000000000000004
slab_full...0x3004B3A8 squeezed....0x0000 full........0x0000
slab_part...0x32D4D000 partial.....0x0001 empty.......0x0000
slab_dead...0x0 dead........0x0000
pile_lock...0x00000000 alloc_lock..0x00000000
heap........0x300000B8
HANDLERS:
```

```
cookie......0x00000000  reconfig....0x00000000
init........0x00000000  free........0x00000000

KDB(0)>
```

# slab subcommand

## Purpose

The **slab** subcommand displays the slab structure at the specified address.

## Syntax

**slab** *address*

## Parameters

- *address* – Specifies the memory address for which you want to display the slab structure. The *address* parameter is required.

The slab command performs some basic error checking on the data structure. If the **slab** subcommand finds an invalid slab ID, a warning message is generated. If the pile to which the slab belongs has an invalid ID, a warning message is generated.

## Aliases

No aliases.

## Example

The following is an example of how to use the **slab** subcommand:

```
KDB(0)> slab 0x337EC000
Pile........0x32BAE600
ID..........0x534C4142 prev........0x32BAE630 next........0x32BAE630
freelist....0x337EC3FC datastart...0x337EC07C objsize.....0x0380
flags.......0x0005     refcount....0x00000001 maxrefcnt...0x00000049
pages.......0x0010     pagesinuse..0x0010

KDB(0)>
```

# Chapter 32. Display NFS information subcommands

The subcommands in this category can be used to display NFS information. These subcommands include the following:

- cupboard
- sockpint
- sockcup
- svcxprt

# cupboard subcommand

## Purpose

The **cupboard** subcommand displays either a list of the current KRPC server cupboards in use or displays the contents of a single KRPC server **cupboard** structure.

## Syntax

**cupboard** [*effectiveaddress*]

## Parameters

- *effectiveaddress* – Specifies the effective address of a **cupboard** structure to display. If this parameter is omitted, a list of the current KRPC server cupboards is displayed.

## Aliases

No aliases.

## Example

The following is an example of how to use the **cupboard** subcommand:

```
KDB(0)> cupboard
3286BE00  rpc.lockd
KDB(0)> cupboard 3286BE00
CUPBOARD............ 3286BE00
RPC Services:
program 100021, Version 4, Dispatch .lm_nlm4_dispatch
program 100021, Version 3, Dispatch .lm_nlm_dispatch
program 100021, Version 2, Dispatch .lm_nlm_dispatch
program 100021, Version 1, Dispatch .lm_nlm_dispatch

Service Handles:
Address     Sockpint
32D4BD00    3286BE00  Master UDP handle - receiving on port 32769
3285D100    3286BE00  Master UDP handle - receiving on port 32788

Manager Section:
cb_mgrlock...... 00000000  cb_event........ FFFFFFFF
cb_all_stop.....    FALSE  cb_wrap.........    FALSE
cb_start_thread.    FALSE
cb_mgr_thread... 00004D9F  cb_mon_thread... 0000429F
cb_svc_thread... 00004B9D  cb_ogre_thread.. 00004EA1
cb_xprt......... 32D4BD00  cb_free_xprt.... 32D4B800
cb_next......... 00000000  cb_name......... rpc.lockd

Count Section:
cb_maxthreads. 00000020  cb_threads. 00000005
cb_active..... 00000000  cb_ideal.... 00000001
cb_idle1...... 00050000  cb_idle5.... 00050000
cb_idle15..... 0004FC08  cb_reserve.. 00000000
cb_threads1... 00050000  cb_threads5. 00050000
cb_threads15.. 0004FC08

Sockcup Section:
cb_sclock....... 00000000  cb_scfree........... 32BE2780
cb_scfirst...... 00000000  cb_sclast........... 00000000
cb_num_sockcups. 000005DC  cb_queued_sockcups.. 00000000
cb_queued1...... 00000000  cb_ququqed5......... 00000000
cb_queued15..... 0000013B

Service Threads Waiting:
Thread Slot    Service Handle
```

```
     65         32D5D300
     64         32D4BF00
     72         32D4B900
     68         32D4BE00
     75         32D4B700   (Main)

Queued Sockcups:
None

KDB(0)>
```

## sockpint subcommand

### Purpose

The **sockpint** subcommand displays the contents of a KRPC server **sockpint** structure.

### Syntax

**sockpint** *effectiveaddress*

### Parameters

- *effectiveaddress* – Specifies the effective address of the **sockpint** structure to display.

### Aliases

No aliases.

### Example

The following is an example of how to use the **sockpint** subcommand:

```
KDB(0)> sockpint 34FFA8C
SOCKPINT............ 0034FFA8C
sp_lock... 0194387B  sp_expand_lock. 12F05400  sp_event.... 20363AF8
sp_xprt... 00067C00  sp_cupboard.... F8505400  sp_socket... 38307EC0
sp_ref.... 1A144800  sp_time........ 00018063
sp_queued... B42B3800
KDB(0)>
```

## sockcup subcommand

### Purpose

The **sockcup** subcommand displays the contents of a KRPC server **sockcup** structure.

### Syntax

**sockcup** *effectiveaddress*

### Parameters

- *effectiveaddress* – Specifies the effective address of the **sockcup** structure to display.

### Aliases

No aliases.

### Example

The following is an example of how to use the **sockcup** subcommand:

```
KDB(0)> sockcup 3D32532
SOCKCUP............ 003D32532
Next.. 0194387B  Mbuf.. 12F05400  Sockpint.. 20363AF8
KDB(0)>
```

# svcxprt subcommand

## Purpose

The **svcxprt** subcommand displays the contents of a KRPC server **svcxprt** structure.

## Syntax

**svcxprt** *effectiveaddress*

## Parameters

- *effectiveaddress* – Specifies the effective address of the **svcxprt** structure to display.

## Aliases

No aliases.

## Example

The following is an example of how to use the **svcxprt** subcommand:

```
KDB(0)> svcxprt 428C82
SVCXPRT............ 00428C82

xp_next....... 0194387B  xp_tid........ 12F05400  xp_flags...... 20363AF8
xp_cb......... 00003800  xp_sp......... 000C3BA0  xp_sock....... 00067C00
xp_ops........ 38307EC0  xp_cred....... 1A144800  xp_type....... B42B3800
xp_sockout.... 00000C80  xp_socksendsz. 0000387B  xp_sockrecvsz. 0F5462C4
xp_p1......... 0020A063  xp_p2......... 00182803  xp_p3......... 00014181
xp_read_dsb... 000038A0  xp_closeproc.. 00084BCF  xp_callouts... 64558375
xp_maxthreads. 0001B005  xp_minthreads. 00064BCC  xp_addrlen.... 00018063
xp_port.......     F850
xp_sockcup.... 000C4BF6  93E56060  00009061
xp_verf....... 02146085  0000A084  00063804
KDB(0)>
```

# Chapter 33. Time subcommands

The subcommands in this category are used to determine the elapsed time from the previous use of the KDB kernel debugger, and to determine Timer Request Block (TRB) information. These subcommands include the following:

- time
- trb

 

## time subcommand

### Purpose

The **time** subcommand determines the elapsed time from the last time the KDB kernel debugger was exited to the time it was entered.

**Note:** The **time** subcommand is only available in the KDB kernel debugger. It is not included in the **kdb** command.

### Syntax

**time**

### Parameters

No parameters.

### Aliases

No aliases.

### Example

The following is an example of how to use the **time** subcommand:

```
KDB(4)> debug ?  //debug help
vmm HW lookup debug... on with arg 'dbg1++', off with arg 'dbg1--'
vmm tr/tv cmd debug... on with arg 'dbg2++', off with arg 'dbg2--'
vmm SW lookup debug... on with arg 'dbg3++', off with arg 'dbg3--'
symbol lookup debug... on with arg 'dbg4++', off with arg 'dbg4--'
stack trace debug..... on with arg 'dbg5++', off with arg 'dbg5--'
BRKPT debug (list).... on with arg 'dbg61++', off with arg 'dbg61--'
BRKPT debug (instr)... on with arg 'dbg62++', off with arg 'dbg62--'
BRKPT debug (suspend). on with arg 'dbg63++', off with arg 'dbg63--'
BRKPT debug (phantom). on with arg 'dbg64++', off with arg 'dbg64--'
BRKPT debug (context). on with arg 'dbg65++', off with arg 'dbg65--'
DABR debug (address).. on with arg 'dbg71++', off with arg 'dbg71--'
DABR debug (register). on with arg 'dbg72++', off with arg 'dbg72--'
DABR debug (status)... on with arg 'dbg73++', off with arg 'dbg73--'
BRAT debug (address).. on with arg 'dbg81++', off with arg 'dbg81--'
BRAT debug (register). on with arg 'dbg82++', off with arg 'dbg82--'
BRAT debug (status)... on with arg 'dbg83++', off with arg 'dbg83--'
BRKPT debug (context). on  //this debug feature is enable
KDB(4)> debug dbg5++  //enable debug mode
stack trace debug..... on
KDB(4)> f  //stack frame in debug mode
thread+000180 STACK:
=== Look for traceback at 0x00015278
=== Got traceback at 0x00015280 (delta = 0x00000008)
=== has_tboff = 1, tb_off = 0xD8
=== Trying to find Stack Update Code from 0x000151A8 to 0x00015278
=== Found 0x9421FFA0 at 0x000151B8
=== Trying to find Stack Restore Code from 0x000151A8 to 0x0001527C
=== Trying to find Registers Save Code from 0x000151A8 to 0x00015278
[00015278]waitproc+0000D0 ()
=== Look for traceback at 0x00015274
=== Got traceback at 0x00015280 (delta = 0x0000000C)
=== has_tboff = 1, tb_off = 0xD8
[00015274]waitproc+0000CC ()
=== Look for traceback at 0x0002F400
=== Got traceback at 0x0002F420 (delta = 0x00000020)
=== has_tboff = 1, tb_off = 0x30
[0002F400]procentry+000010 (??, ??, ??, ??)
```

```
/# ls  //Invoke command from command line that calls open
Breakpoint
0024FDE8     stwu   stkp,FFFFFFB0(stkp) stkp=2FF3B3C0,FFFFFFB0(stkp)=2FF3B370
KDB(0)> time  //Report time from leaving the debugger till the break
Command: time  Aliases:
Elapsed time since last leaving the debugger:
2 seconds and 121211136 nanoseconds.
KDB(0)>
```

# trb subcommand

## Purpose

The **trb** subcommand displays Timer Request Block (TRB) information.

## Syntax

**trb** [ ***** | *cpu x*] [*option*]

## Parameters

- ***** – Displays Timer Request Block (TRB) information for TRBs on all processors. Summary information is displayed for some options. To see detailed information, select a specific processor and option.
- *cpu x* – Is the text `cpu` followed by the processor number. It displays TRB information for the specified processor.

  **Note:** The characters *cpu* must be included in the input. The value *x* is the hexadecimal number of the processor.
- *option* – Specifies the option number that indicates the data to be displayed. The available option numbers can be viewed by entering the **trb** subcommand with no arguments.

If this subcommand is entered without parameters, a menu displays that allows you to select the data you want to display.

## Aliases

**timer**

## Example

The following is an example of how to use the **trb** subcommand:

```
KDB(4)> trb  //timer request block subcommand usage
Usage: trb [CPU selector] [1-9]
 CPU selector is '*' for all CPUs, 'cpu n' for CPU n, default is current CPU

Timer Request Block Information Menu
  1. TRB Maintenance Structure - Routine Addresses
  2. System TRB
  3. Thread Specified TRB
  4. Current Thread TRB's
  5. Address Specified TRB
  6. Active TRB Chain
  7. Free TRB Chain
  8. Clock Interrupt Handler Information
  9. Current System Time - System Timer Constants
Please enter an option number:  //<CR/LF>
KDB(4)> trb * 6  //print all active timer request blocks

CPU #0 Active List
          CPU  PRI       ID    SECS     NSECS     DATA FUNC
05689080 0000 0005 FFFFFFFE 00003BBA 23C3B080 05689080 sys_timer+000000
05689600 0000 0003 FFFFFFFE 00003BBA 27DAC680 00000000 pffastsched+000000
05689580 0000 0003 FFFFFFFE 00003BBA 2911BD80 00000000 pfslowsched+000000
0B05A600 0000 0005 00001751 00003BBA 2ADBC480 0B05A618 rtsleep_end+000000
05689500 0000 0003 FFFFFFFE 00003BBB 23186B00 00000000 if_slowsched+000000
0B05A480 0000 0003 FFFFFFFE 00003BBF 2D5B4980 00000000 01B633F0

CPU #1 Active List
          CPU  PRI       ID    SECS     NSECS     DATA FUNC
05689100 0001 0005 FFFFFFFE 00003BBA 23C38E80 05689100 sys_timer+000000
```

```
CPU #2 Active List
          CPU  PRI       ID    SECS     NSECS     DATA FUNC
05689180 0002 0005 FFFFFFFE 00003BBA 23C37380 05689180 sys_timer+000000
0B05A500 0002 0005 00001525 00003BE6 0CFF9500 0B05A518 rtsleep_end+000000

CPU #3 Active List
          CPU  PRI       ID    SECS     NSECS     DATA FUNC
05689200 0003 0005 FFFFFFFE 00003BBA 23C39F80 05689200 sys_timer+000000
(4)> more (^C to quit) ?  //continue
05689880 0003 0005 00000003 00003BBB 01B73180 00000000 sched_timer_post+000000
0B05A580 0003 0005 00000001 00003BBB 0BCA7300 0000000E interval_end+000000

CPU #4 Active List
          CPU  PRI       ID    SECS     NSECS     DATA FUNC
05689280 0004 0005 FFFFFFFE 00003BBA 23C3A980 05689280 sys_timer+000000

CPU #5 Active List
          CPU  PRI       ID    SECS     NSECS     DATA FUNC
05689300 0005 0005 FFFFFFFE 00003BBA 23C39800 05689300 sys_timer+000000
05689780 0005 0005 FFFFFFFF 00003BBF 1B052C00 05C62C40 01ADD6FC

CPU #6 Active List
          CPU  PRI       ID    SECS     NSECS     DATA FUNC
05689380 0006 0005 FFFFFFFE 00003BBA 23C3C200 05689380 sys_timer+000000

CPU #7 Active List
          CPU  PRI       ID    SECS     NSECS     DATA FUNC
05689400 0007 0005 FFFFFFFE 00003BBA 23C38180 05689400 sys_timer+000000
05689680 0007 0003 FFFFFFFE 00003BBA 2DDD3480 00000000 threadtimer+000000
KDB(4)> trb cpu 1 6  //print active list of processor 1
CPU #1 TRB #1 on Active List
 Timer address......................05689100
 trb->to_next.......................00000000
 trb->knext.........................00000000
 trb->kprev.........................00000000
 Owner id (-1 for dev drv)..........FFFFFFFE
 Owning processor...................00000001
 Timer flags........................00000013   PENDING ACTIVE INCINTERVAL
 trb->timerid.......................00000000
 trb->eventlist.....................FFFFFFFF
 trb->timeout.it_interval.tv_sec....00000000
 trb->timeout.it_interval.tv_nsec...00000000
 Next scheduled timeout (secs)......00003BBA
 Next scheduled timeout (nanosecs)..23C38E80
 Completion handler.................000B3BA4  sys_timer+000000
 Completion handler data............05689100
 Int. priority .....................00000005
 Timeout function...................00000000  00000000
 KDB(4)>
```

# Chapter 34. System trace, dump and error log subcommands

The subcommands in this category support some fundamental AIX Reliability and Serviceability features. These subcommands display data in the kernel trace buffers, data in the trace buffers, unprocessed system error log entries, and data in a system memory dump. These subcommands include the following:

- trace
- trcstart
- trcstop
- cdt
- errpt
- mtrace
- check

## trace subcommand

### Purpose

The **trace** subcommand displays data in the kernel trace buffers or data in the trace buffers collected using the “trcstart subcommand” on page 380.

### Syntax

**trace** [**-h**] [*hook*[:*subhook*]]... [#*data*]... [**-c** *channel*]

**trace -K** [**-j** *event1*, *eventN* **-k** *event1*, *eventN*]

### Parameters

- **-h** – Displays trace headers.
- *hook* – Specifies the hexadecimal value of the hook IDs on which to report.
- *:subhook* – Specifies subhooks, if needed. The subhooks are specified as hexadecimal values.

  **Note:** If subhooks are used, the complete syntax must include both the hook and subhook IDs separated by a colon. For example, assume a trace of hook `1d1`, subhook `2d` is desired, the complete hook specification would be `1d1:2d`.
- *data* – Identifies the trace entries you want to display. These entries are hexadecimal values.
- **-c** *channel* – Selects the trace channel for which the contents are to be monitored. The value for *channel* must be a decimal constant in the range 0 to 7. If no channel is specified, a prompt is displayed.
- **-K** – Displays the trace gathered using the **trcstart** subcommand. Trace hooks are displayed in reverse order.
- **-j** *event1 eventN* – Displays trace data only for the events in the list.
- **-k** *event1 eventN* – Displays trace data for the events that are not in the list.

Data is entered into these buffers using the **trace** shell subcommand. If the shell subcommand was not invoked prior to using the **trace** subcommand, the trace buffers are empty.

The **trace** subcommand is not meant to replace the shell **trcrpt** subcommand in *AIX 5L Version 5.3 Technical Reference: Base Operating System and Extensions Volume 2*, which formats the data in more detail. The **trace** subcommand is a facility for viewing system trace data in the event of a system crash before the data is written to disk.

### Aliases

No aliases.

### Example

The following is an example of how to use the **trace** subcommand:

```
KDB(0)> trcstart
Kernel Trace initialiized successfully
Quit out of kdb, for tracing to continue
KDB(0)> q
Debugger entered via keyboard.
.waitproc_find_run_queue+00009C       li    r3,0                <0000000000000000> r3=0000000000000040
KDB(0)> trcstop
Kernel trace stopped successfully
KDB(0)> trace -K
Current entry is #1522 of 1522 at F100009E1460D088
   Hook ID: KERN_SLIH (00000102)    Hook Type: 0
   ThreadIdent: 0000A00B
```

```
   Subhook ID/HookData: 0000
   Data Length: 0008 bytes
   D0: 0049BDF0
Current entry is #1521 of 1522 at F100009E1460D068
   Hook ID: KERN (00000100)    Hook Type: Timestamped 8000
   ThreadIdent: 0000A00B
   Subhook ID/HookData: 0005
   Data Length: 0008 bytes
   D0: 00028B10
Current entry is #1520 of 1522 at F100009E1460D050
   Hook ID: KERN_SLIH (00000102)    Hook Type: 0
   ThreadIdent: 00008009
   Subhook ID/HookData: 0000
   Data Length: 0008 bytes
   D0: 0049BDF0
(0)> more (^C to quit) ?
Current entry is #1519 of 1522 at F100009E1460D038
   Hook ID: KERN_SLIH (00000102)    Hook Type: 0
   ThreadIdent: 00006007
   Subhook ID/HookData: 0000
   Data Length: 0008 bytes
   D0: 0049BDF0
Current entry is #1518 of 1522 at F100009E1460D018
   Hook ID: KERN (00000100)    Hook Type: Timestamped 8000
   ThreadIdent: 00008009
   Subhook ID/HookData: 0005
   Data Length: 0008 bytes
   D0: 00028BB8
Current entry is #1517 of 1522 at F100009E1460CFF8
   Hook ID: KERN (00000100)    Hook Type: Timestamped 8000
   ThreadIdent: 00006007
   Subhook ID/HookData: 0005
   Data Length: 0008 bytes
   D0: 00028BC0
Current entry is #1516 of 1522 at F100009E1460CFB8
```

## trcstart subcommand

### Purpose

The **trcstart** subcommand starts system trace for the KDB kernel debugger. This command cannot be used with the **kdb** command. For more information and to see an example, see “trace subcommand” on page 378.

### Syntax

**trcstart** [ **-f** | **-l** ] [ **-j** *events* ] [ **-k** *events* ] [ **-p** ]

### Parameters

- **-f** – Logs only the first trace buffers collected.
- **-l** – Logs only the last trace buffers collected.
- **-j** *events* – Traces only the specified events. The events must be separated by commas.
- **-k** *events* – Traces events that are not specified. The events must be separated by commas.
- **-p** – Places the processor identifier in each trace event. This parameter can only be used for 64-bit kernels.

The trace daemon starts a system trace. When the trace is viewed with the **trace** subcommand, the most recently-gathered data is shown. The **-l** parameter is the default.

### Aliases

No aliases.

### Example

To trace hooks 101 and 104, use the following subcommand:

```
trcstart -j 101,104
```

## trcstop subcommand

### Purpose

The **trcstop** subcommand stops a kdb trace. This command cannot be used with the **kdb** command. For more information and to see an example, see "trace subcommand" on page 378.

### Syntax

**trcstop**

### Parameters

No parameters.

### Aliases

No aliases.

### Example

See "trace subcommand" on page 378.

## mtrace subcommand

### Purpose

The **mtrace** subcommand displays information about the Lightweight Memory Trace (LMT).

### Syntax

**mtrace** [ **-c** *cpuid* [ **-t rare** | **common** ] | **-d** *addr size* ]

### Parameters

- **–c** *cpuid* - Specifies the logical ID of a processor in decimal format.
- **–d** *addr size* - Specifies the memory trace buffer address and size.
- **–t rare** | **common** - Specifies the type of buffer.

If LMT is in disabled mode, only general LMT information can be displayed. If the **kdb** command is invoked on a live kernel, trace events in buffers cannot be displayed.

If no options are specified, the **mtrace** command displays general information about LMT (the contents of the **mtrc** structure).

If the **-c** and **-t** parameters are specified, trace events recorded in the rare or common memory trace buffer of the specified processor are displayed, with the most recent events displayed first.

If the **-d** parameter is specified, trace events recorded in the buffer at the specified address and of the specified size are displayed. Use the **-d** parameter to display memory trace events saved in the dmp_minimal area of a system dump.

### Aliases

**mtrc**

### Example

The following is an example of the output displayed by the alias **mtrc** subcommand:

```
KDB(0)> mtrc    // display LMT information
MTRC  @ 00000000011732B8
mt_magic.......... ....mtrc
mt_state.......... 00000000  ENABLED
mt_flags.......... 00000000
mt_lock .........@ 0FFFFFFFFFFFC160  00000000
mt_bufsize...[COM] 0000000000098000
mt_bufsize...[RAR] 0000000000065000
mt_reqbufsize[COM] FFFFFFFFFFFFFFFF
mt_reqbufsize[RAR] FFFFFFFFFFFFFFFF
mt_cdtsize........ 00000000007E8278
mt_cdt...........@ F100080010546000
mt_wait........... FFFFFFFFFFFFFFFF

KDB(0)> mtrc -c 0    // display memory trace buffer information of cpu 0
MTRC [COM]   @  F10008000FF99040
mtq_start... F100011870000000
mtq_size.... 0000000000098000
mtq_inptr... F100011870064090

MTRC [RAR]   @  F10008000FF99060
mtq_start... F100011896666000
mtq_size.... 0000000000065000
mtq_inptr... F100011896666630
```

```
KDB(0)> mtrc -c 0 common  // display trace event of common buffer of cpu 0
Display content of buffer: mtrcq @ F10008000FF99040
Current entry at @ F100011870064088
   Hook ID: KERN_SLIH (00000102)    Hook Type:
   ThreadIdent: 00000205
   Subhook ID/HookData: 0000
   Data Length: 0008 bytes
   D0: 0000000003EC2050  ................

Current entry at @ F100011870064068
   Hook ID: KERN_FLIH (00000100)    Hook Type: Timestamped
   ThreadIdent: 00000205
   Subhook ID/HookData: 0005
   Data Length: 0028 bytes
   D0: 000000000002E36C  ................
   D1: 0000000000000000  ................
   D2: F00000002FF47600  ................
   D3: 0000000000000000  ................
   D4: 0000000000000000  ................
```

# cdt subcommand

## Purpose

The **cdt** subcommand displays data in a system memory dump.

**Note:** This subcommand is only available within the **kdb** command. It is not included in the KDB kernel debugger.

## Syntax

**cdt** [**-d**] [*index*] [*entry*]

## Parameters

- **-d** – Indicates that the dump routines in the **/usr/lib/ras/dmprtns** directory are used to display data from component dump tables.
- *index* – Indicates the component dump table to be viewed. This must be a decimal value.
- *entry* – Indicates the data area of the indicated component to be viewed. This must be a decimal value.

Any component dump area can be displayed. With no parameters, all component dump table headers are displayed. If an index is specified, the component dump table header and associated entries are displayed. If both an index and an entry are specified, the data for the indicated area is displayed in both hexadecimal and ASCII. If the **-d** flag is specified, the dump formatting routines, if any, for the specified component are invoked to format the data in the component data areas.

## Aliases

No aliases.

## Example

The following is an example of how to use the **cdt** subcommand:

```
(0)> cdt
1) CDT head name proc, len 001D80E8, entries 96676
2) CDT head name thrd, len 003ABE4C, entries 192489
3) CDT head name errlg, len 00000054, entries 3
4) CDT head name bos, len 00000040, entries 2
5) CDT head name vmm, len 000003D8, entries 30
6) CDT head name sscsidd, len 0000007C, entries 5
7) CDT head name dptSR, len 00000054, entries 3
8) CDT head name scdisk, len 00000130, entries 14
9) CDT head name lvm, len 00000040, entries 2
10) CDT head name SSAGS, len 000000A4, entries 7
11) CDT head name SSAES, len 00000054, entries 3
12) CDT head name ssagateway, len 0000007C, entries 5
13) CDT head name tty, len 00000068, entries 4
14) CDT head name sio_dd, len 00000054, entries 3
15) CDT head name netstat, len 000000E0, entries 10
16) CDT head name ent2104x, len 00000054, entries 3
17) CDT head name cstokdd, len 0000007C, entries 5
18) CDT head name atm_dd_charm, len 00000040, entries 2
19) CDT head name ssadisk, len 000002AC, entries 33
20) CDT head name SSADS, len 00000040, entries 2
21) CDT head name osi_frame, len 0000002C, entries 1
(0)> cdt 12
12) CDT head name ssagateway, len 0000007C, entries 5
CDT    1 name          HashTbl addr 0000000001A25CF0, len 00000040
CDT    2 name         CfgdAdap addr 0000000001A0E044, len 00000004
CDT    3 name         OpenAdap addr 0000000001A0E048, len 00000004
CDT    4 name         LockWord addr 0000000001A0E04C, len 00000004
CDT    5 name             ssa0 addr 0000000001A2D000, len 00000B88
```

```
(0)> cdt -d 12 4
12) CDT head name ssagateway, len 0000007C, entries 5
CDT    4 name         LockWord addr 0000000001A0E04C, len 00000004
01A0E04C: FFFFFFFF                                   ....
```

## errpt subcommand

### Purpose

The **errpt** command displays unprocessed system error log entries.

### Syntax

**errpt**

### Parameters

The **errpt** subcommand displays system error log entries that were not processed by the error daemon. The entries are displayed in ascending chronological order with the oldest first.

### Aliases

No aliases.

### Example

The following is an example of how to use the **errpt** subcommand:

```
KDB(6)> errpt
ERRORS NOT READ BY ERRDEMON (ORDERED CHRONOLOGICALLY):

Error Record:
erec_flags .............        0
erec_len ...............       40
erec_timestamp ......... 4034EA04
erec_rec_len ...........       20
erec_dupcount ..........        0
erec_duptime1 ..........        0
erec_duptime2 ..........        0
erec_rec.error_id ....... 2BFA76F6
erec_rec.resource_name .. SYSPROC
00000000 00000000 00000000            ............

Error Record:
erec_flags .............        0
erec_len ...............      834
erec_timestamp ......... 4036203C
erec_rec_len ...........      814
erec_dupcount ..........        0
erec_duptime1 ..........        0
erec_duptime2 ..........        0
erec_rec.error_id ....... BFE4C025
erec_rec.resource_name .. sysplanar0
01440000 0000003A C6008401 14123700  .D.....:......7.
20040220 00000000 00000000 00000000   .. ...........
000020FF 00200001 00000000 00000000  .. .. ..........
49424D00 55312E31 302D5031 2D433200  IBM.U1.10-P1-C2.
00020000 00000000 00000000 00000000  ................
00000000 00000000 00000000 00000000  ................
00000000 00000000 00000000 00000000  ................
00000000 00000000 00000000 00000000  ................
00000000 00000000 00000000 00000000  ................
00000000 00000000 00000000 00000000  ................
00000000 00000000 00000000 00000000  ................
00000000 00000000 00000000 00000000  ................
00000000 00000000 00000000 00000000  ................
00000000 00000000 00000000 00000000  ................
<snip>
```

## check subcommand

### Purpose

The **check** subcommand runs consistency checkers on kernel data structures.

### Syntax

**check**

**check ?** | **-?**

**check -h** *CheckerName*

**check** [ **-v** ] [ **-l** *level* ] [ **-n** *count* ] *CheckerName* [ **.***SuffixName* ] [ *EffectiveAddress* ]

**check -e** [ **-v** ] [ **-l** *level* ] *CheckerName* [ **.***SuffixName* ] *EffectiveAddress*

### Parameters

- *CheckerName* – Specifies the name of the checker to run. Run the **check** command with no parameters to display the list of known checkers.
- *SuffixName* – Specifies which of the suffixes of the given checker to run. Run the **check** command with the **-h** parameter to display the list of known suffixes for a given checker.
- *Effective Address* – Specifies the effective address of the element to be validated or the effective address of the first element to be validated for lists. Symbols, hexadecimal values, or hexadecimal expressions can be used to specify the effective address.
- **-e** – Specifies that only one element should be checked. This is equivalent to **-n 1**. An effective address must be specified if the **-e** parameter is specified.
- **-h** – Displays help for each suffix of the specified checker.
- **-l** *level* – Specifies the checking level the checker should use. This is a decimal value between 0 and 9. A value of 9 specifies the most detailed checking level and a value of 0 specifies no checking. The default value is 3 (light level) unless the **-e** flag is specified, in which case the default value is 7 (detailed level).
- **-n** *count* – Specifies the number of elements (*count* is a decimal value) to validate.
- **-v** – Specifies that the checker should run in verbose mode and display additional information if the checker supports this option.

### Aliases

No aliases.

### Example

1. To display the list of known checkers, type the following:

   ```
   check
   ```

   Output similar to the following displays:

   ```
   Please specify a checker name:

   Kernel Checkers         Description
   --------------------------------------------------------------------------------
   proc                    Validate proc and pvproc structures
   thread                  Validate thread and pvthread structures

   Kernext Checkers        Description
   --------------------------------------------------------------------------------
   ```

2. To display detailed help for a specified checker, type the following:

```
check -h proc
```

Output similar to the following displays:

```
Checker 'proc' is used to validate pvproc and proc structures:
proc                   check the global pvproc process table
proc <addr>            check a single pvproc
proc.pv_db <addr>      check a list of pvproc linked by pv_dbnext
proc.pv_sched <addr>   check a list of pvproc linked by pv_sched_next/back
proc.pv_siblings <addr> check a list of pvproc linked by pv_siblings
proc.pv_pgrp <addr>    check a list of pvproc linked by pv_pgrpl/pv_pgrpb
proc.pv_ttyl <addr>    check a list of pvproc linked by pv_ttyl
proc.pv_crid <addr>    check a list of pvproc linked by pv_cridnext
For each element, both pvproc and associated proc structure are validated
        <addr> shoud be the address of a pvproc structure (not a proc structure)
```

3. To run proc checker to validate the entire process table, type the following:

   ```
   check -l 7 proc
   ```

   Output similar to the following displays. In this example, a corruption is found in a flag.

   ```
   Corruption found in pvproc.pv_flag: F100020E0000A400+0100 | RASCHK_BAD_BITMASK | Invalid flags
   ```

4. To run proc checker to perform a detailed check on a single process, type the following:

   ```
   check -e -l 7 proc pvproc+006800
   ```

5. To run proc checker to validate the first five elements of a list of processes linked by the pv_siblings field starting at pvproc+00AC00 in verbose mode, type the following:

   ```
   check -l 7 -n 5 -v proc.pv_siblings pvproc+00AC00
   ```

   Output similar to the following displays:

   ```
   Last element checked: F100020E0000AC00 <pvproc+00AC00>
   Last element checked: F100020E0000C000 <pvproc+00C000>
   Last element checked: F100020E0000BC00 <pvproc+00A400>
   Corruption found in pvproc.pv_flag: F100020E0000A400+0100 | RASCHK_BAD_BITMASK | Invalid flags
   Last element checked: F100020E0000B000 <pvproc+00BC00>
   Last element checked: F100020E0000B000 <pvproc+00B000>
   ```

# Chapter 35. Lock subcommands

The subcommands in this category can be used to display information about locks and to check the system for deadlocks. These subcommands include the following:

- lk
- slk
- clk
- dlk
- dla

 

## lk, slk, clk, and dlk subcommands

### Purpose

The **lk** (display lock_t lock), **slk** (display simple lock), **clk** (display complex lock) and **dlk** (display dist lock) subcommands can be used to display information about locks.

**Note:** The **dlk** subcommand is only available with the 64-bit kernel.

### Syntax

**lk** [ *lock_address* ]

**slk** [**-q**] [ *lock_address* ]

**clk** [**-q**] [ *lock_address* ]

**dlk** [**-q**] [ *lock_address* ]

### Parameters

- *lock_address* – Specifies the address of the lock. Symbols, hexadecimal values, and hexadecimal expressions can be used to specify the address.
- **-q** – Keeps instrumentation information from displaying. If instrumentation is set at boot time and the **-q** option is not entered, **slk**, **clk**, and **dlk** show instrumentation information.

If no parameter is given, a default list of locks is displayed.

### Aliases

No aliases.

### Example

Instrumentation is set to on by using the **-L** option of the **bosboot** command. The following is an example of how to use the **lk**, **slk**, **clk** and **dlk** subcommands with instrumentation set to on:

```
KDB(0)> lk                    //show status of default list of locks
Major Locks:
acct_lock                Available
03E6B180
 lock F100109E0866D280 INTERLOCK
 cpu_owner............... 00000000 @ F100109E0866D280
audit_lock               Available
audit_q_lock             Available
audit_w_lock             Available
03BC50F8                 Available
bio_lock                 Available
bus_reg_lock             Available
cio_lock                 Available
clist_lock               Available
cons_lock                Available
core_lock                Available
cred_alloc_lock          Available
cs_lock                  Available
ctrace_lock              Available
devswlock
 lock F100109E0802AF30
 thread_owner............ 0802AF30 @ pvthread+7802A00
dil_lock                 Available
(0)> more (^C to quit) ? ^C          //interrupt
KDB(0)> lk acct_lock               //show lock_t lock acct_lock
```

```
acct_lock Available
KDB(0)> nm acct_lock              //show address of acct_lock
Symbol Address : 0149CF00
   TOC Address : 0149A2D0
KDB(0)> lk 0149CF00               //show acct_lock using address
acct_lock Available
KDB(0)> slk cio_lock              //show simple lock cio_lock
cio_lock Available
Instrumented lock...... @ F100109E0801A0E0
................lockname: FFFFFFFF
KDB(0)> slk -q cio_lock              //show cio_lock without instrumentation
cio_lock Available
KDB(0)> clk jfs_quota_lock          //show complex lock jfs_quota_lock
jfs_quota_lock Available
Instrumented lock...... @ F100109E0C006EA0
................lockname: FFFFFFFF
KDB(0)> clk -q jfs_quota_lock          //show jfs_quota_lock without instrumentation
jfs_quota_lock Available
KDB(0)> dlk wlm_classes_lock           //show dist lock wlm_classes_lock
wlm_classes_lock
 mutex.............. F100109E0C000050 write owner ........ 0000000000000000
 writer await........ FFFFFFFFFFFFFFFF count.............. 0000000000000000
 writer wait reader.. FFFFFFFFFFFFFFFF count.............. 0000000000000000
 reader await........ FFFFFFFFFFFFFFFF count.............. 0000000000000000
 readers active...... 0000000000000000 reader counter.....@ F10010F004056080
 node interlace...... 0000000000000200 instrumented.......@ F100109E08017ED0
 cpg shift......................... 00 cpu groups........................ 02
 grp shift......................... 01 grp mask.......................... 01
Group counters:
SRAD ID: 0000
 Group 00........... 0000000000000000 @ F10010F004056080
 Group 01........... 0000000000000000 @ F10010F004056100
Instrumented lock...... @ F100109E08017ED0
................lockname: 00000000
KDB(0)> dlk -q wlm_classes_lock          //show wlm_classes_lock without instrumentation
wlm_classes_lock
 mutex.............. F100109E0C000050 write owner ........ 0000000000000000
 writer await........ FFFFFFFFFFFFFFFF count.............. 0000000000000000
 writer wait reader.. FFFFFFFFFFFFFFFF count.............. 0000000000000000
 reader await........ FFFFFFFFFFFFFFFF count.............. 0000000000000000
 readers active...... 0000000000000000 reader counter.....@ F10010F004056080
 node interlace...... 0000000000000200 instrumented.......@ F100109E08017ED0
 cpg shift......................... 00 cpu groups........................ 02
 grp shift......................... 01 grp mask.......................... 01
Group counters:
SRAD ID: 0000
 Group 00........... 0000000000000000 @ F10010F004056080
 Group 01........... 0000000000000000 @ F10010F004056100
```

## dla subcommand

### Purpose

The **dla** subcommand checks the system for deadlocks and displays details about threads waiting for locks.

**Note:** The **dla** subcommand is only available with the **kdb** command.

### Syntax

**dla** [ { **-p** [*cpu*] } | *tid* ]

### Parameters

- **-p** – Reports only on the locks waited on by the specified processor. If no processor is specified, reports on all of the processors.
- *cpu* – Specifies the cpu number.
- *tid* – Report on locks waited on by the thread specified by this thread identifier.

If no arguments are given, the **dla** subcommand analyzes the system for deadlocks. The **dla** subcommand also shows details on any thread waiting for a lock.

### Aliases

No aliases.

### Example

The following is an example of how to use the **dla** subcommand:

```
(0)> dla

No deadlock, but chain from tid 42C5, that waits for the first line lock,
owned by Owner-Id that waits for the next line lock, and so on ...
     LOCK NAME    |       ADDRESS      |  OWNER-ID | LOCK STATUS | WAITING FUNC
      ptrace_lock | 0x00000000006E9898 | Tid  1B37 |  0x20000000 | slock_ppc

No deadlock, but chain from tid 53AF, that waits for the first line lock,
owned by Owner-Id that waits for the next line lock, and so on ...
     LOCK NAME    |       ADDRESS      |  OWNER-ID | LOCK STATUS | WAITING FUNC
      ptrace_lock | 0x00000000006E9898 | Tid  1B37 |  0x20000000 | slock_ppc
No deadlock found
(0)> dla 42C5

No deadlock, but chain from tid 42C5, that waits for the first line lock,
owned by Owner-Id that waits for the next line lock, and so on ...
     LOCK NAME    |       ADDRESS      |  OWNER-ID | LOCK STATUS | WAITING FUNC
      ptrace_lock | 0x00000000006E9898 | Tid  1B37 |  0x20000000 | slock_ppc
No locks waited on for thread EA002100
(0)> dla -p 0
No locks being waited on for processor 0
(0)> dla -p
No deadlock found
```

# Chapter 36. Network subcommands

The subcommands in this category are used to print network information. These subcommands include the following:

- ifnet
- tcb
- udb
- sock
- tcpcb
- mbuf
- netm
- sockinfo
- ndd
- nsdbg
- netstat
- route
- rtentry
- rxnode

# ifnet subcommand

## Purpose

The **ifnet** subcommand prints network interface information.

## Syntax

**ifnet** [*slot* | *effectiveaddress*]

## Parameters

- *slot* – Specifies the slot number within the **ifnet** table for which data is to be displayed. This value must be a decimal number.
- *effectiveaddress* – Specifies the effective address of an **ifnet** entry to display.

If no parameter is specified, information is displayed for each entry in the **ifnet** table. Display data for individual entries by specifying either a slot number or by specifying the address of the entry.

## Aliases

No aliases.

## Example

The following is an example of how to use the **ifnet** subcommand:

```
KDB(0)> ifnet 1
SLOT  1 ---- IFNET INFO ----(@ F10006000CDF2000)----
    name........ en0      unit........ 00000000 mtu......... 000005DC
    flags....... 5E080863
        (UP|BROADCAST|NOTRAILERS|RUNNING|SIMPLEX|NOECHO|BPF|GROUP_ROUTING...
...|64BIT|CHECKSUM_OFFLOAD|PSEG|CANTCHANGE|MULTICAST)
    timer....... 00000000 metric...... 00000000

            address: 9.53.85.113         dest address: 9.53.85.255
            netmask: 255.255.255.0       bk-ptr: F10006000CDF2000
            rtentry: 0   ifa_flags: 1
            ifa_refcnt: 5        ifa_rtrequest: 0

    init()...... 00000000 output().... 03DE2160 start()..... 00000000
    done()...... 00000000 ioctl()..... 03DE2178 reset()..... 00000000
    watchdog().. 00000000 ipackets.... 00000376 ierrors..... 00000000
    opackets.... 00000247 oerrors..... 00000000 collisions.. 00000000
    next........@000000002C0F8F8      addrlen............. 00000006
    type........ 00000006 (ETHER)
    hdrlen...... 0000000E index....... 00000002
    lastchange.. 40B36BE3 sec 00030003 usec

    ibytes...... 00048FDC obytes...... 0001BD0C imcasts..... 00000000
    omcasts..... 00000007 iqdrops..... 00000000 noproto..... 00000000
    baudrate.... 00A00000 arpdrops.... 0000000000000000
    ifbufminsize 00000000 devno....... 00000000 chan........ 00000000
    multiaddrs..@F100061000BFF068      tapctl.....@0000000000000000
    tap()....... 00000000 arpres().... 03DE2190 arprev().... 03DE21A8
    arpinput().. 03DE21C0 ifq_head....@0000000000000000
    ifq_tail....@0000000000000000      ifq_len..... 0000000000000000
    ifq_maxlen.. 0000000000000000      ifq_drops... 00000000
    ifq_slock... 0000000000000000      slock....... 0000000000000000
    multi_lock.. 0000000000000000      6_multi_lock 0000000000000000
    addrlist_lck 0000000000000000      gidlist..... @0000000000000000
    ip6tomcast() 03DE21D8 ndp_bcopy(). 03DE21F0 ndp_bcmp().. 03DE2200
    ndtype...... 02032800 multiaddrs6.@0000000000000000
```

```
    vipaxface..@0000000000000000

KDB(0)>
```

## tcb subcommand

## Purpose

The **tcb** subcommand displays the **inpcb** structure for TCP connections.

## Syntax

**tcb** [**-s** | **-b** *index* | *effectiveaddress*]

## Parameters

- **-s** – Displays a one line summary of every tcb entry.
- **-b** *index* – Specifies the bucket number within the tcb hash table. All tcb entries in this bucket are displayed in detail. The **-b** indicates that the number that follows is a bucket number and not an effective address.
- *effectiveaddress* – Specifies the effective address of a tcb entry to display in detail.

If no parameters are specified, detailed information is displayed for all entries in the tcb table. A summary of all entries or detailed information for a specific entry can be displayed with the appropriate parameters.

## Aliases

No aliases.

## Example

The following is an example of how to use the **tcb** subcommand:

```
KDB(0)> tcb -s
SLOT 13  TCB --------- INPCB  INFO ----(@ F100061000BF5A58)----
SLOT 21  TCB --------- INPCB  INFO ----(@ F100061000BF7258)----
SLOT 23  TCB --------- INPCB  INFO ----(@ F100061000BF7A58)----
SLOT 25  TCB --------- INPCB  INFO ----(@ F1000610004C0A58)----
SLOT 37  TCB --------- INPCB  INFO ----(@ F100061000BF2258)----
SLOT 111  TCB --------- INPCB  INFO ----(@ F10006100039BA58)----
SLOT 512  TCB --------- INPCB  INFO ----(@ F100061000BF5258)----
SLOT 513  TCB --------- INPCB  INFO ----(@ F100061000BF6A58)----
SLOT 514  TCB --------- INPCB  INFO ----(@ F100061000BF6258)----
SLOT 6864  TCB --------- INPCB  INFO ----(@ F100061002D84258)----
SLOT 8269  TCB --------- INPCB  INFO ----(@ F1000610003F6258)----
SLOT 8288  TCB --------- INPCB  INFO ----(@ F1000610003F6A58)----
SLOT 8289  TCB --------- INPCB  INFO ----(@ F100061000C1AA58)----
SLOT 9090  TCB --------- INPCB  INFO ----(@ F100061000BF2A58)----
KDB(0)> tcb F100061000BF2258 //tcb address in slot 37
SLOT 37  TCB --------- INPCB  INFO ----(@ F100061000BF2258)----
    next........@0000000000000000  prev........@0000000000000000
    head........@0000000003E63780  faddr_6.....@F100061000BF2278
    iflowinfo... 00000000 fport....... 00000000 fatype...... 00000000
    oflowinfo... 00000000 lport....... 00000025 latype...... 00000000
    laddr_6.....@F100061000BF2290  socket......@F100061000BF2000
    ppcb........@F100061000BF2360  route_6.....@F100061000BF22B0
    ifa.........@0000000000000000  flags....... 00000400
    proto....... 00000000 tos......... 00000000 ttl......... 0000003C
    rcvttl...... 00000000 rcvif.......@0000000000000000
    options.....@0000000000000000  refcnt...... 00000000
    lock........ 0000000000000000  rc_lock..... 0000000000000000
    moptions....@0000000000000000  hash.next...@F10006000C6D6378
    hash.prev...@F10006000C6D6378  timewait.nxt@0000000000000000
    timewait.prv@0000000000000000  inp_v6opts  @0000000000000000
    inp_pmtu....@0000000000000000

---- SOCKET INFO ----(@ F100061000BF2000)----
    type........ 0001 (STREAM)
```

```
    opts........ 0006 (ACCEPTCONN|REUSEADDR)
    linger...... 0000 state....... 0080 (PRIV)
    pcb.....@F100061000BF2258  proto...@0000000003E5A7A8
    lock....@F1000610002D7640  head....@0000000000000000
    q0......@0000000000000000  q.......@0000000000000000
    q0len....... 0000 qlen........ 0000 qlimit...... 03E8
    timeo....... 0000 error....... 0000 special..... 0A08
    pgid.... 0000000000000000  oobmark. 0000000000000000

snd:cc...... 0000000000000000  hiwat... 0000000000004000
    mbcnt... 0000000000000000  mbmax... 0000000000010000
    lowat... 0000000000001000  mb......@0000000000000000
    sel.....@0000000000000000  events...... 0000
    iodone.. 00000000          ioargs..@0000000000000000
    lastpkt.@0000000000000000  wakeone. FFFFFFFFFFFFFFFF
    timer...@0000000000000000  timeo... 00000000
    flags....... 0000 ()
    wakeup.. 00000000          wakearg.@0000000000000000
    lockwtg. FFFFFFFFFFFFFFFF

MBUF LIST

rcv:cc...... 0000000000000000  hiwat... 0000000000004000
    mbcnt... 0000000000000000  mbmax... 0000000000010000
    lowat... 0000000000000001  mb......@0000000000000000
    sel.....@0000000000000000  events...... 0001
    iodone.. 00000000          ioargs..@0000000000000000
    lastpkt.@0000000000000000  wakeone. FFFFFFFFFFFFFFFF
    timer...@0000000000000000  timeo... 00000000
    flags....... 0008 (SEL|NOTIFY)
    wakeup.. 00000000          wakearg.@0000000000000000
    lockwtg. FFFFFFFFFFFFFFFF

MBUF LIST

    tpcb....@0000000000000000  fdev_ch.@F10006000C3E16C0
    sec_info@0000000000000000  qos.....@0000000000000000
    gidlist.@0000000000000000  private.@0000000000000000
    uid..... 00000000 bufsize. 00000000 threadcnt00000000
    nextfree@0000000000000000
    siguid.. 00000000 sigeuid. 00000000 sigpriv. 00000000
    sndtime. 0000000000000000  sec  0000000000000000  usec
    rcvtime. 0000000000000000  sec  0000000000000000  usec
    saioq...@0000000000000000  saioqhd.@0000000000000000
    accept.. FFFFFFFFFFFFFFFF  frcatime 00000000
    isnoflgs 00000000 ()
    rcvlen.. 0000000000000000  frcaback@0000000000000000
    frcassoc@0000000000000000  frcabckt 0000000000000000
    iodone.. 00000000          iodonefl 00000000 ()
    ioarg...@0000000000000000  refcnt.. 0000000000000000

proc/fd:  29/19
KDB(0)>
```

## udb subcommand

### Purpose

The **udb** subcommand displays the inpcb structure for UDP connections.

### Syntax

**udb** [**-s** | **-b** *index* | *effectiveaddress*]

### Parameters

- **-s** – Displays a one line summary of every udb entry.
- **-b** *index* – Specifies the bucket number within the udb hash table. All udb entries in this bucket are displayed in detail. The **-b** indicates that the number that follows is a bucket number and not an effective address.
- *effectiveaddress* – Specifies the effective address of a udb entry to display in detail.

If no parameters are specified, detailed information is displayed for all entries in the udb table. Display a summary of all entries or detailed information for a specific entry by specifying the appropriate parameters.

### Aliases

No aliases.

### Example

The following is an example of how to use the **udb** subcommand:

```
KDB(0)> udb -s
SLOT 13  UDB --------- INPCB  INFO ----(@ F100061000BF3000)----
SLOT 37  UDB --------- INPCB  INFO ----(@ F100061000BF3200)----
SLOT 111  UDB --------- INPCB  INFO ----(@ F100061000BFB600)----
SLOT 123  UDB --------- INPCB  INFO ----(@ F10006100039D600)----
SLOT 123  UDB --------- INPCB  INFO ----(@ F10006100039DE00)----
SLOT 123  UDB --------- INPCB  INFO ----(@ F10006100039D800)----
SLOT 135  UDB --------- INPCB  INFO ----(@ F100061000410A00)----
SLOT 514  UDB --------- INPCB  INFO ----(@ F100061000BFF800)----
SLOT 518  UDB --------- INPCB  INFO ----(@ F100061000BFBC00)----
KDB(0)> udb F100061000BFB600  //udb address in slot 111
SLOT 111  UDB --------- INPCB  INFO ----(@ F100061000BFB600)----
    next........@0000000000000000  prev........@0000000000000000
    head........@0000000003E63888  faddr_6.....@F100061000BFB620
    iflowinfo... 00000000 fport....... 00000000 fatype...... 00000000
    oflowinfo... 00000000 lport....... 0000006F latype...... 00000000
    laddr_6.....@F100061000BFB638  socket......@F1000610002DC400
    ppcb........@0000000000000000  route_6.....@F100061000BFB658
    ifa.........@0000000000000000  flags....... 00000400
    proto....... 00000000 tos......... 00000000 ttl......... 0000001E
    rcvttl...... 00000000 rcvif.......@0000000000000000
    options.....@0000000000000000  refcnt...... 00000000
    lock........ 0000000000000000  rc_lock..... 0000000000000000
    moptions....@0000000000000000  hash.next...@F10006000CA64A68
    hash.prev...@F10006000CA64A68  timewait.nxt@0000000000000000
    timewait.prv@0000000000000000  inp_v6opts  @0000000000000000
    inp_pmtu....@0000000000000000

---- SOCKET INFO ----(@ F1000610002DC400)----
    type........ 0002 (DGRAM)
    opts........ 0104 (REUSEADDR|OOBINLINE)
    linger...... 0000 state....... 0100 (NBIO)
    pcb.....@F100061000BFB600  proto...@0000000003E5A738
    lock....@F1000610002D7280  head....@0000000000000000
    q0......@0000000000000000  q.......@0000000000000000
```

```
    q0len....... 0000 qlen........ 0000 qlimit...... 0000
    timeo....... 0000 error....... 0000 special..... 088C
    pgid.... 0000000000000000  oobmark. 0000000000000000

snd:cc...... 0000000000000000  hiwat... 0000000000002400
    mbcnt... 0000000000000000  mbmax... 0000000000009000
    lowat... 0000000000001000  mb......@0000000000000000
    sel.....@0000000000000000  events...... 0000
    iodone.. 00000000          ioargs..@0000000000000000
    lastpkt.@0000000000000000  wakeone. FFFFFFFFFFFFFFFF
    timer...@0000000000000000  timeo... 00000000
    flags....... 0048 (SEL|NOINTR|INHERIT|NOTIFY)
    wakeup.. 03C59490          wakearg.@F100061000BFED18
    lockwtg. FFFFFFFFFFFFFFFF

MBUF LIST

rcv:cc...... 0000000000000000  hiwat... 000000000000A460
    mbcnt... 0000000000000000  mbmax... 0000000000029180
    lowat... 0000000000000001  mb......@0000000000000000
    sel.....@0000000000000000  events...... 0000
    iodone.. 00000000          ioargs..@0000000000000000
    lastpkt.@F10006100039D000  wakeone. FFFFFFFFFFFFFFFF
    timer...@0000000000000000  timeo... 00000000
    flags....... 0048 (SEL|NOINTR|INHERIT|NOTIFY)
    wakeup.. 03C594A8          wakearg.@F100061000BFEC00
    lockwtg. FFFFFFFFFFFFFFFF

MBUF LIST

    tpcb....@0000000000000000  fdev_ch.@0000000000000000
    sec_info@0000000000000000  qos.....@0000000000000000
    gidlist.@0000000000000000  private.@0000000000000000
    uid..... 00000000 bufsize. 00000000 threadcnt00000000
    nextfree@0000000000000000
    siguid.. 00000000 sigeuid. 00000000 sigpriv. 00000000
    sndtime. 0000000000000000  sec  0000000000000000  usec
    rcvtime. 0000000000000000  sec  0000000000000000  usec
    saioq...@0000000000000000  saioqhd.@0000000000000000
    accept.. FFFFFFFFFFFFFFFF  frcatime 00000000
    isnoflgs 00000000 ()
    rcvlen.. 0000000000000000  frcaback@0000000000000000
    frcassoc@0000000000000000  frcabckt 0000000000000000
    iodone.. 00000000          iodonefl 00000000 ()
    ioarg...@0000000000000000  refcnt.. 0000000000000000

proc/fd:
KDB(0)>
```

## sock subcommand

### Purpose

The **sock** subcommand prints socket structure for UDP and TCP sockets

### Syntax

**sock** [**-d**] [**tcp** | **udp**] [*effectiveaddress*]

**sock -s** [**tcp** | **udp**]

**sock -f**

### Parameters

- **-d** – Suppresses the display of send and receive buffer information for a socket.
- **-s** – Displays a one-line summary of every socket. If the optional **tcp** or **udp** parameter is used with **-s**, displays a summary of only the specified socket types.
- **-f**
- **tcp** – Displays socket information for TCP blocks only.
- **udp** – Displays socket information for UDP blocks only.
- *effectiveaddress* – Specifies the effective address of a particular socket structure to display.

If no parameter is specified, detailed information is displayed for every allocated TCP or UDP socket on the system. The displayed information can be restricted to only a particular socket type by using the **tcp** parameter or the **udp** parameter. Specifying the effective address of a particular **socket** structure, limits the display to that structure.

### Aliases

No aliases.

### Example

The following is an example of how to use the **sock** subcommand:

```
KDB(0)> sock -s tcp
--- TCP (inpcb: @ F1000610003F0258) --- SOCKET  @ F1000610003F0000
--- TCP (inpcb: @ F1000610003F1A58) --- SOCKET  @ F1000610003F1800
--- TCP (inpcb: @ F1000610003F2258) --- SOCKET  @ F1000610003F2000
--- TCP (inpcb: @ F100061002A6DA58) --- SOCKET  @ F100061002A6D800
--- TCP (inpcb: @ F1000610003F0A58) --- SOCKET  @ F1000610003F0800
--- TCP (inpcb: @ F100061000435A58) --- SOCKET  @ F100061000435800
--- TCP (inpcb: @ F1000610003FBA58) --- SOCKET  @ F1000610003FB800
--- TCP (inpcb: @ F1000610003F2A58) --- SOCKET  @ F1000610003F2800
--- TCP (inpcb: @ F1000610003EE258) --- SOCKET  @ F1000610003EE000
--- TCP (inpcb: @ F100061002AE0258) --- SOCKET  @ F100061002AE0000
--- TCP (inpcb: @ F100061002AE0A58) --- SOCKET  @ F100061002AE0800
--- TCP (inpcb: @ F100061002AD1A58) --- SOCKET  @ F100061002AD1800
--- TCP (inpcb: @ F100061002A6D258) --- SOCKET  @ F100061002A6D000
--- TCP (inpcb: @ F10006100035CA58) --- SOCKET  @ F10006100035C800
--- TCP (inpcb: @ F100061000343258) --- SOCKET  @ F100061000343000
--- TCP (inpcb: @ F100061000435258) --- SOCKET  @ F100061000435000
--- TCP (inpcb: @ F100061000437A58) --- SOCKET  @ F100061000437800
--- TCP (inpcb: @ F1000610003F1258) --- SOCKET  @ F1000610003F1000
KDB(0)> sock F1000610003F0000 first socket address from above
---- SOCKET INFO ----(@ F1000610003F0000)----
    type........ 0001 (STREAM)
    opts........ 0006 (ACCEPTCONN|REUSEADDR)
    linger...... 0000 state....... 0080 (PRIV)
```

```
    pcb.....@F1000610003F0258  proto...@0000000003E427A8
    lock....@F1000610003FF600  head....@0000000000000000
    q0......@0000000000000000  q.......@0000000000000000
    q0len....... 0000 qlen........ 0000 qlimit...... 03E8
    timeo....... 0000 error....... 0000 special..... 0A08
    pgid.... 0000000000000000  oobmark. 0000000000000000

snd:cc...... 0000000000000000  hiwat... 000000000000E000
    mbcnt... 0000000000000000  mbmax... 0000000000038000
    lowat... 0000000000001000  mb......@0000000000000000
    sel.....@0000000000000000  events...... 0000
    iodone.. 00000000          ioargs..@0000000000000000
    lastpkt.@0000000000000000  wakeone. FFFFFFFFFFFFFFFF
    timer...@0000000000000000  timeo... 00000000
    flags....... 0000 ()
    wakeup.. 00000000          wakearg.@0000000000000000
    lockwtg. FFFFFFFFFFFFFFFF

MBUF LIST

rcv:cc...... 0000000000000000  hiwat... 000000000000E000
    mbcnt... 0000000000000000  mbmax... 0000000000038000
    lowat... 0000000000000001  mb......@0000000000000000
    sel.....@0000000000000000  events...... 0001
    iodone.. 00000000          ioargs..@0000000000000000
    lastpkt.@0000000000000000  wakeone. FFFFFFFFFFFFFFFF
    timer...@0000000000000000  timeo... 00000000
    flags....... 0008 (SEL|NOTIFY)
    wakeup.. 00000000          wakearg.@0000000000000000
    lockwtg. FFFFFFFFFFFFFFFF

MBUF LIST

    tpcb....@0000000000000000  fdev_ch.@F10006000CE0F600
    sec_info@0000000000000000  qos.....@0000000000000000
    gidlist.@0000000000000000  private.@0000000000000000
    uid..... 00000000 bufsize. 00000000 threadcnt00000000
    nextfree@0000000000000000
    siguid.. 00000000 sigeuid. 00000000 sigpriv. 00000000
    sndtime. 0000000000000000  sec  0000000000000000  usec
    rcvtime. 0000000000000000  sec  0000000000000000  usec
    saioq...@0000000000000000  saioqhd.@0000000000000000
    accept.. FFFFFFFFFFFFFFFF  frcatime 00000000
    isnoflgs 00000000 ()
    rcvlen.. 0000000000000000  frcaback@0000000000000000
    frcassoc@0000000000000000  frcabckt 0000000000000000
    iodone.. 00000000          iodonefl 00000000 ()
    ioarg...@0000000000000000  refcnt.. 0000000000000000

proc/fd:  98/19
KDB(0)>
```

## tcpcb subcommand

## Purpose

The **tcpcb** subcommand displays the **tcpcb** structure.

## Syntax

**tcpcb** [**-s** | *effectiveaddress*]

## Parameters

- **-s** – Displays a one-line summary of every tcb entry.
- *effectiveaddress* – Specifies the effective address of a **tcpcb** structure to be displayed. Use symbols, hexadecimal values, or hexadecimal expressions to specify the address.

If no parameter is specified, detailed information is displayed for all **tcpcb** structures. A single **tcpcb** structure is displayed by specifying the effective address of the structure, and a summary of all **tcpcb** structures is displayed by using the **-s** option.

## Aliases

No aliases.

## Example

The following is an example of how to use the **tcpcb** subcommand:

```
KDB(0)> tcpcb -s
---- TCP ----(inpcb: @ F1000610003F5258)----
---- TCPCB ----(@ F1000610003F5360)----
---- TCP ----(inpcb: @ F1000610003F2A58)----
---- TCPCB ----(@ F1000610003F2B60)----
---- TCP ----(inpcb: @ F1000610003F3258)----
---- TCPCB ----(@ F1000610003F3360)----
---- TCP ----(inpcb: @ F100061002A8E258)----
---- TCPCB ----(@ F100061002A8E360)----
---- TCP ----(inpcb: @ F1000610003F5A58)----
---- TCPCB ----(@ F1000610003F5B60)----
---- TCP ----(inpcb: @ F100061000395A58)----
---- TCPCB ----(@ F100061000395B60)----
---- TCP ----(inpcb: @ F1000610003F4A58)----
---- TCPCB ----(@ F1000610003F4B60)----
---- TCP ----(inpcb: @ F1000610003F4258)----
---- TCPCB ----(@ F1000610003F4360)----
---- TCP ----(inpcb: @ F1000610003F3A58)----
---- TCPCB ----(@ F1000610003F3B60)----
---- TCP ----(inpcb: @ F100061000387258)----
---- TCPCB ----(@ F100061000387360)----
---- TCP ----(inpcb: @ F10006100046F258)----
---- TCPCB ----(@ F10006100046F360)----
---- TCP ----(inpcb: @ F100061002A8EA58)----
---- TCPCB ----(@ F100061002A8EB60)----
---- TCP ----(inpcb: @ F1000610003EE258)----
---- TCPCB ----(@ F1000610003EE360)----
---- TCP ----(inpcb: @ F1000610002C9A58)----
---- TCPCB ----(@ F1000610002C9B60)----
---- TCP ----(inpcb: @ F10006100049F258)----
---- TCPCB ----(@ F10006100049F360)----
KDB(0)> tcpcb F1000610003F5360  //address of the first tcpcb structure from above
---- TCP ----(inpcb: @ F1000610003F5258)----
---- TCPCB ----(@ F1000610003F5360)----
    seg_next......@F1000610003F5360  seg_prev......@F1000610003F5360
    t_softerror... 00000000 t_state....... 00000001 (LISTEN)
```

```
    t_timer....... 00000000 (TCPT_REXMT)
    t_timer....... 00000000 (TCPT_PERSIST)
    t_timer....... 00000000 (TCPT_KEEP)
    t_timer....... 00000000 (TCPT_2MSL)
    t_rxtshift.... 00000000 t_rxtcur...... 00000006 t_dupacks..... 00000000
    t_maxseg...... 00000200 t_force....... 00000000
    t_flags....... 00000020 (RFC1323|COPYFLAGS)
    t_oobflags.... 00000000 ()
    t_template....@0000000000000000  t_inpcb.......@F1000610003F5258
    t_iobc........ 00000000 t_timestamp... 014C0801 snd_una....... 00000000
    snd_nxt....... 00000000 snd_up........ 00000000 snd_wl1....... 00000000
    snd_wl2....... 00000000 iss........... 00000000
    snd_wnd....... 0000000000000000 rcv_wnd....... 0000000000000000
    rcv_nxt....... 00000000 rcv_up........ 00000000 irs........... 00000000
    snd_wnd_scale. 00000000 rcv_wnd_scale. 00000000 req_scale_sent 00000000
    req_scale_rcvd 00000000 last_ack_sent. 00000000 timestamp_rec. 00000000
    timestamp_age. 00000000 rcv_adv....... 00000000 snd_max....... 00000000
    snd_cwnd...... 000000003FFFC000        snd_ssthresh.. 000000003FFFC000
    t_idle........ 00000000 t_rtt......... 00000000 t_rtseq....... 00000000
    t_srtt........ 00000000 t_rttvar...... 00000006 t_rttmin...... 00000002
    max_rcvd...... 0000000000000000        max_sndwnd.... 0000000000000000
    t_peermaxseg.. 00000200 snd_in_pipe... 00000000
    sack_data.....@0000000000000000         snd_recover... 00000000
    snd_high...... 00000000 snd_ecn_max... 00000000 snd_ecn_clear. 00000000
    t_splice_with.@0000000000000000         t_splice_flags 00000000

KDB(0)>
```

## mbuf subcommand

### Purpose

The **mbuf** subcommand displays mbuf information.

### Syntax

**mbuf** [**-p** | [**-a**] [**-n**] [**-d**]] [*effectiveaddress*]

### Parameters

- **-p** – Displays the private **mbuf** structure pool information.
- **-a** – Follows the packet chain. The *effectiveaddress* parameter is required for this flag.
- **-n** – Follows the **mbuf** structure chain within a packet. The *effectiveaddress* parameter is required for this flag.
- **-d** – Suppresses printing of the **mbuf** structure data and displays only the **mbuf** structure header. This is helpful when only the **mbuf** structure header information is required. The *effectiveaddress* parameter is required for this flag.
- *effectiveaddress* – Specifies the effective address of an **mbuf** structure to be displayed. Use symbols, hexadecimal values, or hexadecimal expressions to specify the address.

Display the packet chain and **mbuf** structure chains within packets by using the **-a** parameter and the **-n** parameter.

### Aliases

No aliases.

### Example

The following is an example of how to use the **mbuf** subcommand:

```
KDB(1)> mbuf -p  total cluster pools............00000001  cluster pool @..........700F8D40
 p_next..................00000000 p_size..................0000000A  p_inuse..................00000001
m_outcnt................00000001  m_maxoutcnt.............00000002 next.....................70168F00
 tail....................70110F00 p_lock..................004A7EE4  p_debug
@..............70EF6600 failed..................00000000  KDB(1)> mbuf 70168F00
 m.......................70168F00  m_next..................00000000 m_nextpkt...............71210F00
 m_data..................71164800 m_len...................00000010 m_type..........
0001 DATA  m_flags......... 0041 (M_EXT|M_EXT2) ext_buf.................71164800
 ext_free................0026C058 ext_size................00000400  ext_arg.................700F8D40
ext_forw................70168F2C  ext_back................70168F2C ext_hasxm...............00000000
 ext_xmemd.....@.........70168F38 ext_debug.....@.........70EF6750
-------------------------------------------------------------------
71164800: 7116 4400  3172 D58C  0000 0000  0000 0000   q.D.1r..........
```

## netm subcommand

### Purpose

The **netm** subcommand displays the **net_malloc** event records that are stored in the kernel.

### Syntax

**netm** [**-c** *display_count*] [**-t** *type* [,type[,...]]] [**-s** *size* [,*size*[,...]]]

**netm -a** [*effectiveaddress*]

**netm -i** *starting_index*

**netm -e** [*outstand_mem*]

### Parameters

- **-c** *display_count* – Specifies how many of the last records of **net_malloc** events you want to display.
- **-a** – Displays all records of the **net_malloc** events.
- **-a***effectiveaddress* – Displays only the **net_malloc** events associated with the specified address.
- **-i** *starting_index* – Displays the **net_malloc** events started from the events record numbered *starting_index*
- **-e** – Displays a list of **net_malloc** memory addresses that are not freed.
- **-e** *outstand_mem* – Displays **net_malloc** events related to the outstanding memory specified by *outstand_mem*.
- **-t** *type* – Limits the display to specified types of blocks. Valid values are a subset of those defined in *INITKMEMNAMES* in the net_malloc.h file.
- **-s** *size* – Limits the display to specified sizes of blocks.

The **netm** subcommand is only available after the **net_malloc_police** attribute is turned on, and the display begins with the latest event. The **netm** subroutine displays up to 16 stack traces in the **net_malloc** event.

### Aliases

No aliases.

### Example

No example.

## sockinfo subcommand

### Purpose

The **sockinfo** subcommand displays several different socket-related structures.

### Syntax

**sockinfo** *effectiveaddress TypeOfAddress*[**-d**]

### Parameters

- *effectiveaddress* – Specifies the effective address of the structure to be displayed.
- *TypeOfAddress* – Identifies the type of structure to which the effective address points. Valid address types are **socket**, **inpcb**, **rawcb**, **unpcb**, **ripcb**and **tcpcb**.
- **-d** – Suppresses the display of send and receive buffer information for a socket.

### Aliases

**si**

### Example

The following is an example of how to use the **sockinfo** subcommand:

```
KDB(0)> sock tcp -s
--- TCP (inpcb: @ F1000610003F0258) --- SOCKET  @ F1000610003F0000
--- TCP (inpcb: @ F1000610003F1A58) --- SOCKET  @ F1000610003F1800
--- TCP (inpcb: @ F1000610003F2258) --- SOCKET  @ F1000610003F2000
--- TCP (inpcb: @ F100061002A6DA58) --- SOCKET  @ F100061002A6D800
--- TCP (inpcb: @ F1000610003F0A58) --- SOCKET  @ F1000610003F0800
--- TCP (inpcb: @ F100061000435A58) --- SOCKET  @ F100061000435800
--- TCP (inpcb: @ F1000610003FBA58) --- SOCKET  @ F1000610003FB800
--- TCP (inpcb: @ F1000610003F2A58) --- SOCKET  @ F1000610003F2800
--- TCP (inpcb: @ F1000610003EE258) --- SOCKET  @ F1000610003EE000
--- TCP (inpcb: @ F100061002AE0258) --- SOCKET  @ F100061002AE0000
--- TCP (inpcb: @ F100061002A6D258) --- SOCKET  @ F100061002A6D000
--- TCP (inpcb: @ F100061002AD1A58) --- SOCKET  @ F100061002AD1800
--- TCP (inpcb: @ F100061000343258) --- SOCKET  @ F100061000343000
--- TCP (inpcb: @ F100061000435258) --- SOCKET  @ F100061000435000
--- TCP (inpcb: @ F100061000437A58) --- SOCKET  @ F100061000437800
--- TCP (inpcb: @ F1000610003F1258) --- SOCKET  @ F1000610003F1000
KDB(0)> sockinfo F1000610003F0258 inpcb address of first inpcb in list above
---- TCPCB ----(@ F1000610003F0360)----
    seg_next......@F1000610003F0360  seg_prev......@F1000610003F0360
    t_softerror... 00000000 t_state....... 00000001 (LISTEN)
    t_timer....... 00000000 (TCPT_REXMT)
    t_timer....... 00000000 (TCPT_PERSIST)
    t_timer....... 00000000 (TCPT_KEEP)
    t_timer....... 00000000 (TCPT_2MSL)
    t_rxtshift.... 00000000 t_rxtcur...... 00000006 t_dupacks..... 00000000
    t_maxseg...... 00000200 t_force....... 00000000
    t_flags....... 00000020 (RFC1323|COPYFLAGS)
    t_oobflags.... 00000000 ()
    t_template....@0000000000000000  t_inpcb.......@F1000610003F0258
    t_iobc........ 00000000 t_timestamp... 6886EC01 snd_una....... 00000000
    snd_nxt....... 00000000 snd_up........ 00000000 snd_wl1....... 00000000
    snd_wl2....... 00000000 iss........... 00000000
    snd_wnd....... 0000000000000000 rcv_wnd....... 0000000000000000
    rcv_nxt....... 00000000 rcv_up........ 00000000 irs........... 00000000
    snd_wnd_scale. 00000000 rcv_wnd_scale. 00000000 req_scale_sent 00000000
    req_scale_rcvd 00000000 last_ack_sent. 00000000 timestamp_rec. 00000000
    timestamp_age. 00000006 rcv_adv....... 00000000 snd_max....... 00000000
    snd_cwnd...... 000000003FFFC000        snd_ssthresh.. 000000003FFFC000
    t_idle........ 00000006 t_rtt......... 00000000 t_rtseq....... 00000000
```

```
    t_srtt........ 00000000 t_rttvar...... 00000006 t_rttmin...... 00000002
    max_rcvd...... 0000000000000000        max_sndwnd.... 0000000000000000
    t_peermaxseg.. 00000200 snd_in_pipe... 00000000
    sack_data.....@0000000000000000         snd_recover... 00000000
    snd_high...... 00000000 snd_ecn_max... 00000000 snd_ecn_clear. 00000000
    t_splice_with.@0000000000000000         t_splice_flags 00000000


-------- TCB --------- INPCB  INFO ----(@ F1000610003F0258)----
    next........@0000000000000000  prev........@0000000000000000
    head........@0000000003E4B780  faddr_6.....@F1000610003F0278
    iflowinfo... 00000000 fport....... 00000000 fatype...... 00000000
    oflowinfo... 00000000 lport....... 0000000D latype...... 00000000
    laddr_6.....@F1000610003F0290  socket......@F1000610003F0000
    ppcb........@F1000610003F0360  route_6.....@F1000610003F02B0
    ifa.........@0000000000000000  flags....... 00000400
    proto....... 00000000 tos......... 00000000 ttl......... 0000003C
    rcvttl...... 00000000 rcvif.......@0000000000000000
    options.....@0000000000000000  refcnt...... 00000000
    lock........ 0000000000000000  rc_lock..... 0000000000000000
    moptions....@0000000000000000  hash.next...@F10006000C6A6138
    hash.prev...@F10006000C6A6138  timewait.nxt@0000000000000000
    timewait.prv@0000000000000000  inp_v6opts  @0000000000000000
    inp_pmtu....@0000000000000000

---- SOCKET INFO ----(@ F1000610003F0000)----
    type........ 0001 (STREAM)
    opts........ 0006 (ACCEPTCONN|REUSEADDR)
    linger...... 0000 state....... 0080 (PRIV)
    pcb.....@F1000610003F0258  proto...@0000000003E427A8
    lock....@F1000610003FF600  head....@0000000000000000
    q0......@0000000000000000  q.......@0000000000000000
    q0len....... 0000 qlen........ 0000 qlimit...... 03E8
    timeo....... 0000 error....... 0000 special..... 0A08
    pgid.... 0000000000000000  oobmark. 0000000000000000

snd:cc...... 0000000000000000  hiwat... 000000000000E000
    mbcnt... 0000000000000000  mbmax... 0000000000038000
    lowat... 0000000000001000  mb......@0000000000000000
    sel.....@0000000000000000  events...... 0000
    iodone.. 00000000          ioargs..@0000000000000000
    lastpkt.@0000000000000000  wakeone. FFFFFFFFFFFFFFFF
    timer...@0000000000000000  timeo... 00000000
    flags....... 0000 ()
    wakeup.. 00000000          wakearg.@0000000000000000
    lockwtg. FFFFFFFFFFFFFFFF

MBUF LIST

rcv:cc...... 0000000000000000  hiwat... 000000000000E000
    mbcnt... 0000000000000000  mbmax... 0000000000038000
    lowat... 0000000000000001  mb......@0000000000000000
    sel.....@0000000000000000  events...... 0001
    iodone.. 00000000          ioargs..@0000000000000000
    lastpkt.@0000000000000000  wakeone. FFFFFFFFFFFFFFFF
    timer...@0000000000000000  timeo... 00000000
    flags....... 0008 (SEL|NOTIFY)
    wakeup.. 00000000          wakearg.@0000000000000000
    lockwtg. FFFFFFFFFFFFFFFF

MBUF LIST

    tpcb....@0000000000000000  fdev_ch.@F10006000CE0F480
    sec_info@0000000000000000  qos.....@0000000000000000
    gidlist.@0000000000000000  private.@0000000000000000
    uid..... 00000000 bufsize. 00000000 threadcnt00000000
    nextfree@0000000000000000
```

```
    siguid.. 00000000 sigeuid. 00000000 sigpriv. 00000000
    sndtime. 0000000000000000  sec  0000000000000000  usec
    rcvtime. 0000000000000000  sec  0000000000000000  usec
    saioq...@0000000000000000  saioqhd.@0000000000000000
    accept.. FFFFFFFFFFFFFFFF  frcatime 00000000
    isnoflgs 00000000 ()
    rcvlen.. 0000000000000000  frcaback@0000000000000000
    frcassoc@0000000000000000  frcabckt 0000000000000000
    iodone.. 00000000          iodonefl 00000000 ()
    ioarg...@0000000000000000  refcnt.. 0000000000000000

proc/fd:  98/19
proc/fd: fd: 19
              SLOT NAME     STATE      PID    PPID          ADSPACE  CL #THS

pvproc+018800   98*inetd    ACTIVE 00620D6 0017056 000000002002D555   0 0001


KDB(0)>
```

## ndd subcommand

### Purpose

The **ndd** subcommand displays the network device driver statistics.

### Syntax

**ndd** [**-s** | *effectiveaddress* | **-n** *nddname*]

### Parameters

- **-s** – Displays the list of all of the valid network device driver tables and gives the address of each **ndd** structure and the name of the corresponding network interface.
- *effectiveaddress* – Specifies the effective address from which the **ndd** structure is read. Use symbols, hexadecimal values, or hexadecimal expressions to specify the address.
- **-n** *nddname* – Indicates a network interface name is used to specify which **ndd** structure is to be read.

When it is used with an address or network interface name, the **ndd** subcommand displays a detailed description of the corresponding table. When it is used with the **-s** parameter, a list of valid network interfaces and the addresses of their **ndd** structures is printed. If no parameters are used, the **ndd** subcommand displays a detailed description of all of the valid network device driver tables.

### Aliases

No aliases.

### Example

The following is an example of how to use the **ndd** subcommand:

```
KDB(0)> ndd -s
  --- NDD ADDR ---(@ F10010E00C69A030)---
      name..... ent1             alias....  en1
  --- NDD ADDR ---(@ F10010E00C6AB030)---
      name..... ent0             alias....  en0
  --- NDD ADDR ---(@ F10010E00BD64028)---
      name..... tok0             alias....  tr0
KDB(0)> ndd -n ent0
    ---- NDD INFO ----(@ F10010E00C6AB030)----
    name............ ent0     alias........... en0
    ndd_next.........@F10010E00BD64028
    flags........... 0063091B
    (UP|BROADCAST|RUNNING|NOECHO|ALT ADDRS|64BIT|CHECKSUM_OFFLOAD|PSEG...
...)
    ndd_open()..... 03D87690 ndd_close().... 03D876C0 ndd_output..... 03D876A8
    ndd_ctl()...... 03D876D8 ndd_stat()..... 03D65A28 receive()...... 03D65A10

    ndd_refcnt....... 00000001          ndd_correlator...@F10010E00C6AB000
    ndd_mtu.......... 000005EA          ndd_mintu........ 0000003C
    ndd_addrlen...... 00000006          ndd_physaddr..... 000255AF36F2
    ndd_hdrlen....... 0000000E
    ndd_type......... 00000007 (802.3 Ethernet)
    ndd_demuxer......@0000000003D65BB8  ndd_nsdemux......@F10010F000340000
    ndd_demuxsource.. 00000000          ndd_specdemux....@F10010F000B77000
    ndd_demux_lock... 0000000000000000  ndd_lock......... 0000000000000000
    ndd_trace........@0000000000000000  ndd_trace_arg....@0000000000000000
    ndd_speclen...... 0000008C          ndd_specstats....@F10010E00C6B7BA0
    ndd_ipackets..... 0000D5E3          ndd_opackets..... 000060FA
    ndd_ierrors...... 00000000          ndd_oerrors...... 00000000
    ndd_ibytes....... 007C0235          ndd_obytes....... 00210113
    ndd_recvintr..... 0000D287          ndd_xmitintr..... 00000002
    ndd_ipackets_drop 00000000          ndd_nobufs....... 00000000
```

```
    ndd_xmitque_max.. 00000004          ndd_xmitque_ovf.. 00000000

KDB(0)>
```

# nsdbg subcommand

## Purpose

The **nsdbg** subcommand displays the ns_alloc and free event records stored in the kernel.

**Note:** This functionality is only available if the *ndd_event_tracing* parameter is turned on by using the **no** command.

## Syntax

**nsdbg** [**-i** *starting_index*] [**-c** *display_count*] [**-n** *nddname*[,*nddname*[,...]] ]

## Parameters

- **-i** *starting_index* – Displays events starting with the event record specified with the *starting_index* parameter.
- **-c** *display_count* – Displays only the events specified with the *display_count* parameter.
- **-n** *nddname* – Displays the events associated with the network interface that have names specified with the *nddname* parameter.

If no parameters are specified, the **nsdbg** subcommand displays all event records stored in the kernel.

## Aliases

No aliases.

## Example

No example.

# netstat subcommand

## Purpose

The **netstat** subcommand symbolically displays the contents of various network-related data structures for active connections.

## Syntax

**netstat** [**-n** ] [**-D**] [**-c**] [**-P**] [**-m** | **-s** | **-ss** | **-u** | **-v**] [ { **-A** **-a** } | { **-r** **-C** **-i** **-I** *Interface* } ]
[ **-f** *AddressFamily* ] [**-p** *Protocol*] [**-Zc** | -**Zi** | **-Zm** | **-Zs**] [*Interval*] [*System*]

## Parameters

- **-n** – Shows network addresses as numbers. When the **-n** flag is not specified, the **netstat** command interprets addresses where possible and displays them symbolically. This flag can be used with any of the display formats.
- **-D** – Shows the number of packets received, transmitted, and dropped in the communications subsystem.
- **-c** – Shows the statistics of the Network Buffer Cache.
- **-P** – Shows the statistics of the Data Link Provider Interface (DLPI).
- **-m** – Shows statistics recorded by the memory management routines.
- **-s** – Shows statistics for each protocol.
- **-ss** – Displays all of the non-zero protocol statistics and provides a concise display.
- **-u** – Displays information about domain sockets.
- **-v** – Shows statistics for CDLI-based communications adapters. This flag causes the **netstat** command to run the statistics commands for the **entstat** subcommand, the **tokstat** subcommand, and the **fddistat** subcommand. No flags are issued to these device driver commands.
- **-A** – Shows the address of any protocol control blocks associated with the sockets. This flag acts with the default display and is used for debugging purposes.
- **-a** – Shows the state of all of the sockets. Without this flag, sockets used by server processes are not shown.
- **-r** – Shows the routing tables. Shows routing statistics when it is used with the **-s** flag.
- **-C** – Shows the routing tables, including the user-configured costs and current costs of each route.
- **-i** – Shows the state of all configured interfaces.
- **-I** *Interface*– Shows the state of all of the configured interfaces specified by the *Interface* variable.
- **-f** *AddressFamily* – Limits reports of statistics or address control blocks to those items specified by the *AddressFamily* variable. The following address families are recognized:
  - inet – Indicates the AF_INET address family
  - inet6 – Indicates the AF_INET6 address family
  - ns – Indicates the AF_NS address family
  - unix – Indicates the AF_UNIX address family
- **-p** *Protocol* – Shows statistics about the value specified for the *Protocol* variable, which is either a name for a protocol or an alias for it. Protocol names and aliases are listed in the `/etc/protocols` file. A null response means that there are no numbers to report. The program report of the value specified for the *Protocol* variable is unknown if there is no statistics routine for it.
- **-Zc** – Clears network buffer cache statistics.
- **-Zi** – Clears interface statistics.
- **-Zm** – Clears network memory allocator statistics.
- **-Zs** – Clears protocol statistics. To clear statistics for a specific protocol, use **-p** *Protocol*. For example, to clear the TCP statistics, type the following on the command line:

```
netstat -Zs -p tcp
```

## Aliases

No aliases.

## Example

The following is an example of how to use the **netstat** subcommand:

```
<0>netstat -r

Route Tree for Protocol Family 2 (Internet):
default            advantis.in.ibm.c  UGc    0     0      en0     -   -
freezer.austin.i   9.184.199.232      UGHMW  0     1      en0     -   1
9.184.192/21       shakti.in.ibm.com  U      20   40546   en0     -   -
mqet2.in.ibm.com   9.184.199.12       UGHMW  0     958    en0     -   1
127/8              localhost          U      2     249    lo0     -   -

Route Tree for Protocol Family 24 (Internet v6):
::1                     ::1           UH     0      0     lo0  16896  -
-------------------------------------------------------------------------------
```

## route subcommand

### Purpose

The **route** subcommand displays the **route**e structure at a given address.

### Syntax

**route** *effectiveaddress*

### Parameters

- *effectiveaddress* – Specifies the effective address of the **route** structure to display.

### Aliases

No aliases.

### Example

The following is an example of how to use the **route** subcommand:

```
# netstat -f inet -n -A
Active Internet connections
PCB/ADDR Proto Recv-Q Send-Q  Local Address      Foreign Address    (state)
715a45e8 tcp4       0      0  9.53.85.113.23     9.53.85.114.50921  ESTABLISHED

# Debugger entered via keyboard.
.waitproc_find_run_queue+000150     beq-    cr7.eq,<.waitproc_find_run_queue+000164>
KDB(0)> tcpcb 715a45e8  //tcpcb address from PCB/ADDR column in netstat
---- TCPCB ----(@ 715A45E8)----
    seg_next...... 715A45E8 seg_prev...... 715A45E8
    t_softerror... 00000000 t_state....... 00000004 (ESTABLISHED)
    t_timer....... 00000000 (TCPT_REXMT)
    t_timer....... 00000000 (TCPT_PERSIST)
    t_timer....... 000037D7 (TCPT_KEEP)
    t_timer....... 00000000 (TCPT_2MSL)
    t_rxtshift.... 00000000 t_rxtcur...... 00000003 t_dupacks..... 00000000
    t_maxseg...... 000005B4 t_force....... 00000000
    t_flags....... 00080000 ()
    t_oobflags.... 00000000 ()
    t_iobc........ 00000000 t_template.... 715A4610 t_inpcb....... 715A4544
    t_timestamp... 0F6B4401 snd_una....... C76DF3FE snd_nxt....... C76DF3FE
    snd_up........ C76DF3FD snd_wl1....... A0AC8F2B snd_wl2....... C76DF3FE
    iss........... C76DEF05 snd_wnd....... 0000E420 rcv_wnd....... 00004470
    rcv_nxt....... A0AC8F2C rcv_up........ A0AC8F2B irs........... A0AC8ED2
    snd_wnd_scale. 00000000 rcv_wnd_scale. 00000000 req_scale_sent 00000000
    req_scale_rcvd 00000000 last_ack_sent. A0AC8F2C timestamp_rec. 00000000
    timestamp_age. 000000C0 rcv_adv....... A0ACD39C snd_max....... C76DF3FE
    snd_cwnd...... 0000EF88 snd_ssthresh.. 3FFFC000 t_idle........ 00000069
    t_rtt......... 00000000 t_rtseq....... C76DF3FD t_srtt........ 00000007
    t_rttvar...... 00000003 t_rttmin...... 00000002 max_rcvd...... 00000000
    max_sndwnd.... 0000E420 t_peermaxseg.. 000005B4 snd_in_pipe... 00000000
    sack_data..... 00000000 snd_recover... 00000000 snd_high...... C76DF3FE
    snd_ecn_max... C76DF3FE snd_ecn_clear. C76DF3FE t_splice_with. 00000000
    t_splice_flags 00000000
KDB(0)> tcb 715A4544  //tcb address from the t_inpcb field
-------- TCB --------- INPCB  INFO ----(@ 715A4544)----
    next........ 00000000 prev........ 00000000 head........ 02576600
    iflowinfo... 00000000 faddr_6... @ 715A4558 fport....... 0000C6E9
    fatype...... 00000001 oflowinfo... 00000000 laddr_6... @ 715A4570
    lport....... 00000017 latype...... 00000001 socket...... 715A4400
    ppcb........ 715A45E8 route_6... @ 715A4588 ifa......... 00000000
    flags....... 00000400 proto....... 00000000 tos......... 00000000
    ttl......... 0000003C rcvttl...... 00000000 rcvif....... 334A6000
    options..... 00000000 refcnt...... 00000000
```

```
    lock........ 00000000 rc_lock..... 00000000 moptions.... 00000000
    hash.next... 32E1CF4C hash.prev... 32E1CF4C
    timewait.nxt 00000000 timewait.prv 00000000
    inp_v6opts   00000000
---- SOCKET INFO ----(@ 715A4400)----
    type........ 0001 (STREAM)
    opts........ 010C (REUSEADDR|KEEPALIVE|OOBINLINE)
    linger...... 0000 state....... 0102 (ISCONNECTED|NBIO)
    pcb..... 715A4544 proto... 02572168 lock.... 701FACA0 head.... 00000000
    q0...... 00000000 q....... 00000000 q0len....... 0000
    qlen........ 0000 qlimit...... 0000 timeo....... 0000
    error....... 0000 special..... 0A8C pgid.... 00000000 oobmark. 00000000
snd:cc...... 00000000 hiwat... 00004000 mbcnt... 00000000 mbmax... 00010000
    lowat... 00003908 mb...... 00000000 sel..... 00000000 events...... 0000
    iodone.. 00000000 ioargs.. 00000000 lastpkt. 709F6700 wakeone. FFFFFFFF
    timer... 00000000 timeo... 00000000 flags....... 0048 (SEL|NOINTR)
    wakeup.. 026A362C wakearg. 715D1890 lockwtg. FFFFFFFF
rcv:cc...... 00000000 hiwat... 00004470 mbcnt... 00000000 mbmax... 000111C0
    lowat... 00000001 mb...... 00000000 sel..... 00000000 events...... 0004
    iodone.. 00000000 ioargs.. 00000000 lastpkt. 715AEB00 wakeone. FFFFFFFF
    timer... 00000000 timeo... 00000000 flags....... 0048 (SEL|NOINTR)
    wakeup.. 026A362C wakearg. 715D1800 lockwtg. FFFFFFFF
    tpcb.... 00000000 fdev_ch. 300736A0 sec_info 00000000 qos..... 00000000
    gidlist. 00000000 private. 00000000 uid..... 00000000 bufsize. 00000000
    threadcnt00000000 nextfree 00000000 siguid.. 00000000 sigeuid. 00000000
    sigpriv. 00000000
    sndtime. 00000000 sec 00000000 usec rcvtime. 00000000 sec 00000000 usec
    saioq... 00000000 saioqhd. 00000000 accept.. FFFFFFFF frcatime 00000000
    isnoflgs 00000000 ()
    rcvlen.. 00000000 frcaback 00000000 frcassoc 00000000 frcabckt 00000000
    iodone.. 00000000 iodonefl 00000000 ()
    ioarg... 00000000 refcnt.. 00000001 proc/fd:  69/0 69/1 69/2
KDB(0)> route 715A4588  //route address from the route_6 field

    Destination.. 9.53.85.114
    .........rtentry@ 715AEE00.........

    rt_nodes[0]......

        rn_mklist @.. 701FA2E0
            rm_b........... FFFFFFC7     rm_unused......
            rm_flags....... 00000005     rm_mklist...... 00000000
            rmu_mask....... 701F51B0
            mask........... 255.255.255.0
            rm_refs........ 00000000

        rn_p @....... 715AED18
        rn_b......... FFFFFFC7 rn_bmask..... 0000
        rn_flags..... 0000000D (NORMAL|ACTIVE|DUP)
        rn_key....... 9.53.85.0/24

        rn_dupedkey @ 00000000
    rt_nodes[1]......

        rn_mklist @.. 00000000
        rn_p @....... 7095D118
        rn_b......... 00000024 rn_bmask..... 0008
        rn_flags..... 00000004 (ACTIVE)
        rn_off....... 00000004
        rn_l @....... 701FCC2C rn_r @....... 7095D518
    gateway...... 9.53.85.113
    rt_redisctime 00000000 rt_refcnt.... 00000003
    rt_flags..... 00000001      (UP)
    ifnet @...... 334A6000 ifaddr @..... 701F5100
    rt_genmask @. 00000000 rt_llinfo @.. 00000000
    rt_rmx (rt_metrics):
         locks ... 00000000 mtu ..... 00000000 hopcount. 00000000
```

```
         expire .. 401FDFCB recvpipe. 00000000 sendpipe. 00000000
         ssthresh. 00000000 rtt ..... 00000000 rttvar .. 00000000
         pksent... 00000031
    rt_gwroute @. 00000000 rt_idle...... 00000000
    ipRouteAge... 00000000 rt_proto @... 00000000
    gidstruct @.. 00000000 rt_lock...... 00000000
    rt_intr...... 00000003 rt_duplist @. 00000000
    rt_lu @...... 00000000 rt_timer..... 00000000
    rt_cost_config 00000000

KDB(0)>
```

## rtentry subcommand

### Purpose

The **rtentry** subcommand displays the **rtentry** structure at a given address.

### Syntax

**rtentry** *effectiveaddress*

### Parameters

- *effectiveaddress* – Specifies the effective address of the **rtentry** structure to display.

### Aliases

No aliases.

### Example

The following is an example of how to use the **rtentry** subcommand:

```
# netstat -f inet -r -A -n
Routing tables
Address  Destination      Gateway           Flags   Refs     Use  If   PMTU Exp Groups

Route tree for Protocol Family 2 (Internet):
701fcc44   (32) 7095d118 : 701fcc5c mk = 70a9f080 {(0), (0) }
7095d118   (33) 715aee18 : 7095d100
715aee18   (36) 701fcc2c : 7095d518
701fcc2c 70a5b100 default         9.53.85.1         UGc       0        0  en0     -   -
         mask (0)  mk = 70a9f080 {(0), (0) }
7095d518   (42) 7095d500 : 727bad18
7095d500 9.0.7.1          9.53.85.1         UGHW      0      628  en0  1500     1
727bad18   (43) 727bad00 : 715aed18
727bad00 9.41.85.44       9.53.85.1         UGHW      0        2  en0     -     1
715aed18   (56) 7095d218 : 715aed00 mk = 701fa2e0 {(56), (0) 0 ffff ff00 }
7095d218   (57) 715aef00 : 7095d200
715aef00 9.53.85.0        9.53.85.113       UHSb      0        0  en0     -   -  =>
715aee00 9.53.85/24       9.53.85.113       U         4       49  en0     -   -
         mask (0) 0 ffff ff00  mk = 701fa2e0 {(56), (0) 0 ffff ff00 }
7095d200 9.53.85.113      127.0.0.1         UGHS      0     1195  lo0     -   -
715aed00 9.53.85.255      9.53.85.113       UHSb      0        1  en0     -   -
7095d100 127/8            127.0.0.1         U         2      831  lo0     -   -
         mask (0) 0 ff00
701fcc5c # Debugger entered via keyboard.
.waitproc_find_run_queue+000048      ori    r3,r8,0            <00000000> r3=ppda,r8=0
KDB(0)> rtentry 727bad00  //rtentry address from Routing Address column in netstat

    .........rtentry@ 727BAD00.........

    rt_nodes[0]......

        rn_mklist @.. 00000000
        rn_p @....... 727BAD18
        rn_b......... FFFFFFFF rn_bmask..... 0000
        rn_flags..... 00000004 (ACTIVE)
        rn_key....... 9.41.85.44
        rn_dupedkey @ 00000000
    rt_nodes[1]......

        rn_mklist @.. 00000000
        rn_p @....... 7095D518
        rn_b......... 0000002B rn_bmask..... 0010
        rn_flags..... 00000004 (ACTIVE)
        rn_off....... 00000005
```

```
        rn_l @....... 727BAD00 rn_r @....... 715AED18
    gateway...... 9.53.85.1
    rt_redisctime 00000000 rt_refcnt.... 00000000
    rt_flags..... 00020007      (UP|GATEWAY|HOST|CLONED)
    ifnet @...... 334A6000 ifaddr @..... 701F5100
    rt_genmask @. 00000000 rt_llinfo @.. 00000000
    rt_rmx (rt_metrics):
         locks ... 00000000 mtu ..... 00000000 hopcount. 00000000
         expire .. 401FE02A recvpipe. 00000000 sendpipe. 00000000
         ssthresh. 00000000 rtt ..... 00000000 rttvar .. 00000000
         pksent... 00000002
    rt_gwroute @. 715AEE00 rt_idle...... 00000000
    ipRouteAge... 00000000 rt_proto @... 7095F4A0
    gidstruct @.. 7095B800 rt_lock...... 00000000
    rt_intr...... 0000000B rt_duplist @. 00000000
    rt_lu @...... 00000000 rt_timer..... 00000000
    rt_cost_config 00000000

KDB(0)>
```

## rxnode subcommand

### Purpose

The **rxnode** subcommand displays information about the **radix_node** structure at a specified address.

### Syntax

**rxnode** *effectiveaddress*

### Parameters

- *effectiveaddress* – Specifies the effective address of the **radix_node** structure.

After displaying the **radix_node** structure, the subcommand presents a menu for interactive traversal of the **radix_node** tree. If the **radix_node** is an intermediate node of the tree, the traversal can follow the parent, left, or right nodes. If the displayed **radix_node** is a leaf node, the traversal can only follow the parent node.

### Aliases

No aliases.

### Example

The following is an example of how to use the **rxnode** subcommand:

```
# netstat -f inet -r -A -n
Routing tables
Address  Destination      Gateway           Flags   Refs     Use  If   PMTU Exp Groups

Route tree for Protocol Family 2 (Internet):
701fcc44   (32) 7095d118 : 701fcc5c mk = 70a9f080 {(0), (0) }
7095d118   (33) 715aee18 : 7095d100
715aee18   (36) 701fcc2c : 7095d518
701fcc2c 70a5b100 default         9.53.85.1         UGc       0       0  en0     -   -
         mask (0)  mk = 70a9f080 {(0), (0) }
7095d518   (42) 7095d500 : 715aed18
7095d500 9.0.7.1          9.53.85.1         UGHW      0     1121  en0     -     2
715aed18   (56) 7095d218 : 715aed00 mk = 701fa2e0 {(56), (0) 0 ffff ff00 }
7095d218   (57) 715aef00 : 7095d200
715aef00 9.53.85.0        9.53.85.113       UHSb      0       0  en0     -   -  =>
715aee00 9.53.85/24       9.53.85.113       U         3      80  en0     -   -
         mask (0) 0 ffff ff00  mk = 701fa2e0 {(56), (0) 0 ffff ff00 }
7095d200 9.53.85.113      127.0.0.1         UGHS      2     2221  lo0     -   -
715aed00 9.53.85.255      9.53.85.113       UHSb      0       1  en0     -   -
7095d100 127/8            127.0.0.1         U         2     1469  lo0     -   -
         mask (0) 0 ff00
701fcc5c # Debugger entered via keyboard.
.waitproc+0000E8      ori    r3,r31,0           <003F3780> r3=0,r31=ppda
KDB(0)> rtentry 7095d200  //rtentry address from Routing Address column in netstat


    .........rtentry@ 7095D200.........

    rt_nodes[0]......

        rn_mklist @.. 00000000
        rn_p @....... 7095D218
        rn_b......... FFFFFFFF rn_bmask..... 0000
        rn_flags..... 00000004 (ACTIVE)
        rn_key....... 9.53.85.113
        rn_dupedkey @ 00000000
    rt_nodes[1]......
```

```
        rn_mklist @.. 00000000
        rn_p @....... 715AED18
        rn_b......... 00000039 rn_bmask..... 0040
        rn_flags..... 00000004 (ACTIVE)
        rn_off....... 00000007
        rn_l @....... 715AEF00 rn_r @....... 7095D200
    gateway...... 127.0.0.1
    rt_redisctime 00000000 rt_refcnt.... 00000002
    rt_flags..... 00000807      (UP|GATEWAY|HOST|STATIC)
    ifnet @...... 011EDB70 ifaddr @..... 7095C000
    rt_genmask @. 00000000 rt_llinfo @.. 00000000
    rt_rmx (rt_metrics):
         locks ... 00000000 mtu ..... 00000000 hopcount. 00000000
         expire .. 401FE69F recvpipe. 00000000 sendpipe. 00000000
         ssthresh. 00000000 rtt ..... 00000000 rttvar .. 00000000
         pksent... 000008AD
    rt_gwroute @. 7095D100 rt_idle...... 00000000
    ipRouteAge... 00000000 rt_proto @... 7095F160
    gidstruct @.. 00000000 rt_lock...... 00000000
    rt_intr...... 00000009 rt_duplist @. 00000000
    rt_lu @...... 00000000 rt_timer..... 00000000
    rt_cost_config 00000000

KDB(0)> rxnode 715AEF00  //radix node address from rn_l; can also use rn_r or rn_p

        rn_mklist @.. 00000000
        rn_p @....... 7095D218
        rn_b......... FFFFFFFF rn_bmask..... 0000
        rn_flags..... 0000000D (NORMAL|ACTIVE|DUP)
        rn_key....... 9.53.85.0
        rn_dupedkey @ 715AEE00
         Traverse radix_node tree :
         parent - 1      quit   - 0
         Enter Choice : 1

        rn_mklist @.. 00000000
        rn_p @....... 715AED18
        rn_b......... 00000039 rn_bmask..... 0040
        rn_flags..... 00000004 (ACTIVE)
        rn_off....... 00000007
        rn_l @....... 715AEF00 rn_r @....... 7095D200
         Traverse radix_node tree :
         parent - 1      rn_r   - 2      rn_l   - 3      quit   - 0
         Enter Choice : 2

        rn_mklist @.. 00000000
        rn_p @....... 7095D218
        rn_b......... FFFFFFFF rn_bmask..... 0000
        rn_flags..... 00000004 (ACTIVE)
        rn_key....... 9.53.85.113
        rn_dupedkey @ 00000000
         Traverse radix_node tree :
         parent - 1      quit   - 0
         Enter Choice : 0


KDB(0)>
```

# Chapter 37. Workload Manager (WLM) subcommands

The subcommands in this category support the WLM functions. These subcommands include the following:

- cla
- rules
- bdev
- bqueue

# cla subcommand

## Purpose

The **cla** subcommand displays Workload Manager (WLM) class statistics and configuration information.

## Syntax

**cla *** [*select#*]

**cla** [*classid*]

## Parameters

- *** – Displays the menu if the *select#* parameter is not specified
- *select#* – Displays the class statistics for the selected number.
  - 1 – CPU for all classes
  - 2 – Mem for all classes
  - 3 – Mem for superclasses
  - 4 – CPU for all classes
  - 5 – Mem for one superclass
  - 6 – BIO use for all classes
  - 7 – BIO use for active classes
  - 8 – BIO use, per-disk, for all classes
  - 9 – Totals for all classes and all resources
- *classid* – Displays configuration information for the specified class identifier.

## Aliases

**class**

## Example

The following is an example of how to use the **cla** subcommand completed by using the menu:

```
KDB(0)> cla *
WLM CLASSes
Select the criteria to display by:
 1) CPU use
 2) MEM use
 3) MEM use over superclasses
 4) Superclasses only
 5) MEM use inside a superclass
 6) BIO use
 7) BIO use (show actives classes for all disks)
 8) BIO use (show classes for all disks)
 9) Total Resources
Enter your choice: 1
  (wlm is ON)
                          TIER  %% MIN SHA SMAX HMAX DES  RAP URAPH URAP URAPL PRI NT         TB    TOTALTB
[  0]:                     Unclassified   0   0   0  -1  100  100 100  100     0    0   194  10  0 0x00000000 0x00000000
[ 64]:                        Unmanaged   0   0   0  -1  100  100 100  100     0    0   194  10  0 0x00000000 0x00000000
[128]:                          Default   0   0   0  -1  100  100 100  100     0    0   194   0  0 0x00000000 0x00000000
[129]:                  Default.Default   0   0   0  -1  100  100 100  100     0    0    97   0  1 0x00000000 0x00000000
[130]:                   Default.Shared   0   0   0  -1  100  100 100  100     0    0    97   0  1 0x00000000 0x00000000
[192]:                           Shared   0   0   0  -1  100  100 100  100     0    0   194   0  0 0x00000000 0x00000000
[193]:                   Shared.Default   0   0   0  -1  100  100 100  100     0    0    97   0  1 0x00000000 0x00000000
[194]:                    Shared.Shared   0   0   0  -1  100  100 100  100     0    0    97   0  1 0x00000000 0x00000000
[256]:                           System   0   1   0  -1  100  100 100  100     0    0   194   0  0 0x000043A8 0x00000000
[257]:                   System.Default   0   1   0  -1  100  100 100  100     0    0    97   0  1 0x000043A8 0x00001DB5
[258]:                    System.Shared   0   0   0  -1  100  100 100  100     0    0    97   0  1 0x00000000 0x00000000
[320]:                            Test1   0   0   0  -1  100  100 100  100     0    0   194   0  0 0x00000000 0x00000000
[321]:                    Test1.Default   0   0   0  -1  100  100 100  100     0    0    97   0  1 0x00000000 0x00000000
[322]:                     Test1.Shared   0   0   0  -1  100  100 100  100     0    0    97   0  1 0x00000000 0x00000000

Display configuration for class 256

KDB(0)> cla 256
System (valid) wlm is DoClassif CpuAcct CpuRegul MemAcct MemRegul BioAcct BioRegul TotalCpuAcct TotalCpuRegul TotalDiskioAcct
 TotalDiskioRegul TotalConnectAcct TotalConnectRegul TotalProcAcct TotalProcRegul TotalThrdAcct TotalThrdRegul
```

```
 TotalLoginAcct TotalLoginRegul
             SLOT TIER ETIER N_ETIER INH USE
   ccb+000800  256    0     0       1  0  52
     admin_Xid = (-1/-1)     assign_Xid = (-1/-1)
        perm_flag = 0x00000000   wlmsched_act = 0x00000000
        nb_subclass =  2        nb_loaded_subclasses =  0
        loaded = 0      rset = 0x0000000000000000
        fixed_target = 1        change_level = 3
       key = 0xDCA6910C4BFFD374
      TIER  %% MIN SHA SMAX HMAX DES  RAP URAPH URAP URAPL SRAP  SRAP              CUM%%              TOTAL    HIGH WATER MARK
cpu:     0   1   0  -1  100  100 100  100     0    0   194    0     0 0x000000000000008A 0x0000000000000BB8 0x0000000000000001
mem:     0   8   1  -1  100  100  99   85     0   76  1023   76    38 0x0000000000000468 0x0000000000000000 0x00000000000072E8
dio:     0   0   0  -1  100  100 100  100     0    0  1023    0     0 0x0000000000000002 0x00000000000002E8 0x0000000000000001
        TOTAL RESOURCES
                   MAX    CONSUM
procs:              -1        52
thds:               -1       129
logs:               -1         4
tcpu:(TB)           -1         -
tdio:(512KB)        -1         -
ctime:(sec)         -1         -
        RESOURCES SPECIFIC DATA
      PRI NT         TB    TOTALTB MEAN       &ACT DELTA
cpu:    0  0 0x000043A8 0x00000000    1 0x02086A44     0
         PAGES     NSTL     LRUPO     COMP   CLIENT    COMPR   REDTHR   WAITLIST            MEMTIME
mem:     29326        0         0    23946        0        0 9223372036854775807 0x00000000 0x00000000003F249D
        CPU ACTIVITY HISTORY
     0<  0   0   0   0   0   0   0   0   0   0   0   0   0   0
     0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
     0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
     0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
pages:29326 - pages*time:0x3F249D
-------------------------------------------------------------------------------
```

Display CPU statistics for all classes

```
KDB(0)> cla * 1
  (wlm is ON)                          TIER  %% MIN SHA SMAX HMAX DES  RAP URAPH URAP URAPL PRI NT         TB     TOTALTB
[  0]:                   Unclassified    0   0   0  -1  100  100 100  100     0    0   194  10  0 0x00000000 0x00000000
[ 64]:                      Unmanaged    0   0   0  -1  100  100 100  100     0    0   194  10  0 0x00000000 0x00000000
[128]:                        Default    0   0   0  -1  100  100 100  100     0    0   194   0  0 0x00000000 0x00000000
[129]:                Default.Default    0   0   0  -1  100  100 100  100     0    0    97   0  6 0x00000000 0x00000000
[130]:                 Default.Shared    0   0   0  -1  100  100 100  100     0    0    97   0  6 0x00000000 0x00000000
[192]:                         Shared    0   0   0  -1  100  100 100  100     0    0   194   0  0 0x00000000 0x00000000
[193]:                 Shared.Default    0   0   0  -1  100  100 100  100     0    0    97   0  6 0x00000000 0x00000000
[194]:                  Shared.Shared    0   0   0  -1  100  100 100  100     0    0    97   0  6 0x00000000 0x00000000
[256]:                         System    0   7   0  -1  100  100 100   86     0   13   194  13  0 0x0000648F 0x00000000
[257]:                 System.Default    0   7   0  -1  100  100 100   86     6   13   103  13  6 0x0000648F 0x00000F1F
[258]:                  System.Shared    0   0   0  -1  100  100 100  100     6    6   103   6  6 0x00000000 0x00000000
[320]:                          Test1    0  58   0   3  100  100  55   -2     0   98   194  98  0 0x000508DA 0x00000000
[321]:                  Test1.Default    0   0   0  -1  100  100 100  100    49   49   146  49  6 0x00000000 0x00000000
[322]:                   Test1.Shared    0   0   0  -1  100  100 100  100    49   49   146  49  6 0x00000000 0x00000000
[323]:                     Test1.Sub1    0  42   0   8  100  100  80    2    49   96   146  96  4 0x0002C44A 0x00004373
[324]:                     Test1.Sub2    0  16   0   2  100  100  20  -18    49  106   146 106  4 0x00024490 0x00003740
[384]:                          Test2    0  34   0   2  100  100  37    4     0   93   194  93  0 0x0000AC61 0x00000000
[385]:                  Test2.Default    0   0   0  -1  100  100 100  100    46   46   143  46  6 0x00000000 0x00000000
[386]:                   Test2.Shared    0   0   0  -1  100  100 100  100    46   46   143  46  6 0x00000000 0x00000000
[387]:                     Test2.Sub1    0  21   0   7  100  100  70   11    46   89   143  89  6 0x0000AC61 0x000012D3
[388]:                     Test2.Sub2    0  13   0   3  100  100  30   -7    46   98   143  98  6 0x00000000 0x00000000
[448]:                          Test3    0   0   0   1  100  100 100  100     0    0   194   0  0 0x00000000 0x00000000
[449]:                  Test3.Default    0   0   0  -1  100  100 100  100     0    0    97   0  6 0x00000000 0x00000000
[450]:                   Test3.Shared    0   0   0  -1  100  100 100  100     0    0    97   0  6 0x00000000 0x00000000
[512]:                          Test4    0   0   0  -1  100  100 100  100     0    0   194   0  0 0x00000000 0x00000000
[513]:                  Test4.Default    0   0   0  -1  100  100 100  100     0    0    97   0  6 0x00000000 0x00000000
[514]:                   Test4.Shared    0   0   0  -1  100  100 100  100     0    0    97   0  6 0x00000000 0x00000000
```

| Column | Description |
|---|---|
| First column | The first column is the class ID. The superclass IDs are a multiple of 64, with subclass IDs following numerically. |
| Second column | The second column is the class name. Subclass names have the format <Supername.Subname>. |
| TIER | The tier number for the class. The tier for a superclass is the supertier. The tier for a subclass is the subtier. |
| %% | The current consumption percentage for the class for the resource being displayed. |
| MIN | The user-defined minimum limit for the class (default is 0). |
| SHA | The user-defined number of shares for the class. (default is -1, unregulated) |
| SMAX | The user-defined soft maximum limit for the class. (default is 100) |
| HMAX | The user-defined hard maximum limit for the class. (default is 100) |
| DES | The desired or target percentage for the class. |
| RAP | The Resource Access Priority for the class. This is a value in the range [-100..100]. |
| URAPH | The highest URAP value the class can attain. |
| URAP | The current URAP value for the class. |
| URAPL | The lowest URAP value the class can attain. |
| PRI | The class priority. |
| NT | Index of next available table entry for CPU consumption samples. |
| TB | Timebase units of consumption for the last second. |
| TOTALTB | Decayed total timebase units of consumption. |

# bdev subcommand

## Purpose

The **bdev** subcommand displays Workload Manager (WLM) I/O statistics for block devices.

## Syntax

**bdev** [**a**] [**c**] [**s**] ***** | **-d** *major minor* | *effectiveaddress*

## Parameters

- **a** – Displays detailed (all) I/O statistics.
- **c** – Displays I/O statistics for each class
- **s** – Displays I/O statistics for each device. This is the default.
- ***** – Displays I/O statistics for all managed devices.
- **-d** – Displays I/O statistics for a device specified by the *major* and *minor* numbers.
- *major* – Specifies the major number. This is a hexadecimal value.
- *minor* – Specifies the minor number. This is a hexadecimal value.
- *effectiveaddress* – Specifies the effective or virtual address for a device with a control block. Symbols, hexadecimal values, or hexadecimal expressions can be used to specify the address.

## Aliases

**wlm_bdev**

## Example

The following is an example of how to use the **bdev** subcommand:

```
Display summary statistics for all devices

KDB(0)> bdev *
33507000: ~ dev: 14,0    in_queue:   0 classes:   0 rq/s:   0 act:   0
33459000: ~ dev: 14,1    in_queue:   0 classes:   0 rq/s:   0 act:   0
334B0000: + dev: 14,2    in_queue:   2 classes:   7 rq/s: 157 act: 100

Description of output (above)

Column Description
1 eaddr of bdev control block
2 status of device
  "-" = uregulated
  "~" = no activity
  "+" = active
3 dev: device major, minor number
4 in_queue: number of requests enqueued
5 classes: number of active classes for the device
6 rq/s: number of requests per second for the device
7 act: the percent active for the device

Display statistics for device with major # 14 and minor # 2

KDB(0)> bdev s -d 14 2
334B0000: + dev: 14,2    in_queue:   2 classes:   7 rq/s: 157 act: 100
 flags               0x00000000      lock                 0x00000000
 next                  00000000      nb_cntrl                   4416
 cntrl                 334B00F0      reguls                 334C14F0
 delayed               32AEE000      in_use                        0
 ev_want_free        0xFFFFFFFF      wbd_active_cntrl              7
 wbd_in_queue                 2      wbd_max_queued                6
 dkstat                32AD5274      prev_dk_time             919418
 &current              334B00CC      &info.wbd_last         334B0024
 &info.wbd_max         334B003C      &info.wbd_av           334B0054
 &info.wbd_total       334B0070
Type      RTHR WTHR RQSTS QUEUE STRVD ACTVT
current      0   672    42    11     0     0
wbd_last     0  2512   157    36     0   100
wbd_max   9280 12192  1172   145     1   100
wbd_av       0  2441   152    33     0   100
```

```
wbd_total 12584 23267266 1460289 112491   578 918479

Examples for BIO statistics using cla command

KDB(0)> cla * 6
  (wlm is ON)                       TIER  %% MIN SHA SMAX HMAX DES   RAP URAPH URAP URAPL
[  0]:               Unclassified    0   0   0  -1  100  100 100   100     0    0   511
[ 64]:                  Unmanaged    0   0   0  -1  100  100 100   100     0    0   511
[128]:                    Default    0   0   0  -1  100  100 100   100     0    0   511
[129]:            Default.Default    0   0   0  -1  100  100 100   100     0    0   255
[130]:             Default.Shared    0   0   0  -1  100  100 100   100     0    0   255
[192]:                     Shared    0   0   0  -1  100  100 100   100     0    0   511
[193]:             Shared.Default    0   0   0  -1  100  100 100   100     0    0   255
[194]:              Shared.Shared    0   0   0  -1  100  100 100   100     0    0   255
[256]:                     System    0   0   0  -1  100  100 100   100     0    0   511
[257]:             System.Default    0   0   0  -1  100  100 100   100     0    0   255
[258]:              System.Shared    0   0   0  -1  100  100 100   100     0    0   255
[320]:                      Test1    0  25   0  -1  100  100 100    58     0  106   511
[321]:              Test1.Default    0  25   0  -1  100  100 100    58    53  106   308
[322]:               Test1.Shared    0   0   0  -1  100  100 100   100    53   53   308
[384]:                      Test2    0   3   0  -1   10  100  10    53     0  119   511
[385]:              Test2.Default    0   3   0  -1  100  100 100    53    59  119   315
[386]:               Test2.Shared    0   0   0  -1  100  100 100   100    59   59   315
[448]:                      Test3    0   3   0  -1   10  100  10    53     0  119   511
[449]:              Test3.Default    0   3   0  -1  100  100 100    53    59  119   315
[450]:               Test3.Shared    0   0   0  -1  100  100 100   100    59   59   315
[512]:                      Test4    1   0   0  -1  100  100 100   100   512  512  1023
[513]:              Test4.Default    0   0   0  -1  100  100 100   100   512  512   767
[514]:               Test4.Shared    0   0   0  -1  100  100 100   100   512  512   767

KDB(0)> cla * 7
  (wlm is ON)                       TIER  %% MIN SHA SMAX HMAX DES   RAP URAPH URAP URAPL   RTHR   WTHR DELAY  VERSION
[320]:                      Test1    0  25   0  -1  100  100 100    58     0  106   511
hdisk1                               0  77   0  -1  100  100 100    12     0  224   511      0      0     0        2
[321]:              Test1.Default    0  25   0  -1  100  100 100    58    53  106   308
hdisk1                               0  77   0  -1  100  100 100    12   112  224   367      0    496     0        1
[384]:                      Test2    0   3   0  -1   10  100  10    53     0  119   511
hdisk1                               0  10   0  -1   10  100  11     0     0  254   511      0      0    39        5
[385]:              Test2.Default    0   3   0  -1  100  100 100    53    59  119   315
hdisk1                               0  10   0  -1  100  100 100     5   127  247   382      0     80    28        1
[448]:                      Test3    0   3   0  -1   10  100  10    53     0  119   511
hdisk1                               0  11   0  -1   10  100  12 -100     0  509   511      0      0    35        4
[449]:              Test3.Default    0   3   0  -1  100  100 100    53    59  119   315
hdisk1                               0  11   0  -1  100  100 100     4   254  376   510      0     96    24        1

KDB(0)> cla * 8
  (wlm is ON)                       TIER  %% MIN SHA SMAX HMAX DES   RAP URAPH URAP URAPL   RTHR   WTHR DELAY  VERSION
[  0]:               Unclassified    0   0   0  -1  100  100 100   100     0    0   511
cd0                                  0   0   0  -1  100  100  10   100     0    0   511      0      0     0        1
hdisk0                               0   0   0  -1  100  100  10   100     0    0   511      0      0     0        1
hdisk1                               0   0   0  -1  100  100  10   100     0    0   511      0      0     0        1
[ 64]:                  Unmanaged    0   0   0  -1  100  100 100   100     0    0   511
cd0                                  0   0   0  -1  100  100  10   100     0    0   511      0      0     0        1
hdisk0                               0   0   0  -1  100  100  10   100     0    0   511      0      0     0        1
hdisk1                               0   0   0  -1  100  100  10   100     0    0   511      0      0     0        1
[128]:                    Default    0   0   0  -1  100  100 100   100     0    0   511
cd0                                  0   0   0  -1  100  100  10   100     0    0   511      0      0     0        2
hdisk0                               0   0   0  -1  100  100  10   100     0    0   511      0      0     0        2
hdisk1                               0   0   0  -1  100  100  10   100     0    0   511      0      0     0        2
[129]:            Default.Default    0   0   0  -1  100  100 100   100     0    0   255
cd0                                  0   0   0  -1  100  100  10   100     0    0   255      0      0     0        1
hdisk0                               0   0   0  -1  100  100  10   100     0    0   255      0      0     0        1
hdisk1                               0   0   0  -1  100  100  10   100     0    0   255      0      0     0        1
[130]:             Default.Shared    0   0   0  -1  100  100 100   100     0    0   255
cd0                                  0   0   0  -1  100  100  10   100     0    0   255      0      0     0        1
hdisk0                               0   0   0  -1  100  100  10   100     0    0   255      0      0     0        1
hdisk1                               0   0   0  -1  100  100  10   100     0    0   255      0      0     0        1
[192]:                     Shared    0   0   0  -1  100  100 100   100     0    0   511
cd0                                  0   0   0  -1  100  100  10   100     0    0   511      0      0     0        2
hdisk0                               0   0   0  -1  100  100  10   100     0    0   511      0      0     0        2
hdisk1                               0   0   0  -1  100  100  10   100     0    0   511      0      0     0        2
[193]:             Shared.Default    0   0   0  -1  100  100 100   100     0    0   255
cd0                                  0   0   0  -1  100  100  10   100     0    0   255      0      0     0        1
hdisk0                               0   0   0  -1  100  100  10   100     0    0   255      0      0     0        1
hdisk1                               0   0   0  -1  100  100  10   100     0    0   255      0      0     0        1
[194]:              Shared.Shared    0   0   0  -1  100  100 100   100     0    0   255
cd0                                  0   0   0  -1  100  100  10   100     0    0   255      0      0     0        1
hdisk0                               0   0   0  -1  100  100  10   100     0    0   255      0      0     0        1
hdisk1                               0   0   0  -1  100  100  10   100     0    0   255      0      0     0        1
[256]:                     System    0   0   0  -1  100  100 100   100     0    0   511
cd0                                  0   0   0  -1  100  100  10   100     0    0   511      0      0     0        2
hdisk0                               0   0   0  -1  100  100  10   100     0    0   511      0      0     0        2
hdisk1                               0   0   0  -1  100  100  10   100     0    0   511      0      0     0        2
```

```
[257]:          System.Default  0   0   0  -1  100   100 100   100    0    0   255
cd0                             0   0   0  -1  100   100  10   100    0    0   255      0      0      0        1
hdisk0                          0   0   0  -1  100   100  10   100    0    0   255      0      0      0        1
hdisk1                          0   0   0  -1  100   100  10   100    0    0   255      0      0      0        1
[258]:           System.Shared  0   0   0  -1  100   100 100   100    0    0   255
cd0                             0   0   0  -1  100   100  10   100    0    0   255      0      0      0        1
hdisk0                          0   0   0  -1  100   100  10   100    0    0   255      0      0      0        1
hdisk1                          0   0   0  -1  100   100  10   100    0    0   255      0      0      0        1
[320]:                   Test1  0  25   0  -1  100   100 100    58    0  106   511
cd0                             0   0   0  -1  100   100  10   100    0    0   511      0      0      0        2
hdisk0                          0   0   0  -1  100   100  10   100    0    0   511      0      0      0        2
hdisk1                          0  77   0  -1  100   100 100    12    0  224   511      0      0      0        2
[321]:           Test1.Default  0  25   0  -1  100   100 100    58   53  106   308
cd0                             0   0   0  -1  100   100  10   100    0    0   255      0      0      0        1
hdisk0                          0   0   0  -1  100   100  10   100    0    0   255      0      0      0        1
hdisk1                          0  77   0  -1  100   100 100    12  112  224   367      0    496      0        1
[322]:            Test1.Shared  0   0   0  -1  100   100 100   100   53   53   308
cd0                             0   0   0  -1  100   100  10   100    0    0   255      0      0      0        1
hdisk0                          0   0   0  -1  100   100  10   100    0    0   255      0      0      0        1
hdisk1                          0   0   0  -1  100   100  10   100  112  112   367      0      0      0        1
[384]:                   Test2  0   3   0  -1   10   100  10    53    0  119   511
cd0                             0   0   0  -1   10   100  10   100    0    0   511      0      0      0        5
hdisk0                          0   0   0  -1   10   100  10   100    0    0   511      0      0      0        5
hdisk1                          0  10   0  -1   10   100  11     0    0  254   511      0      0     39        5
[385]:           Test2.Default  0   3   0  -1  100   100 100    53   59  119   315
cd0                             0   0   0  -1  100   100  10   100    0    0   255      0      0      0        1
hdisk0                          0   0   0  -1  100   100  10   100    0    0   255      0      0      0        1
hdisk1                          0  10   0  -1  100   100 100     5  127  247   382      0     80     28        1
[386]:            Test2.Shared  0   0   0  -1  100   100 100   100   59   59   315
cd0                             0   0   0  -1  100   100  10   100    0    0   255      0      0      0        1
hdisk0                          0   0   0  -1  100   100  10   100    0    0   255      0      0      0        1
hdisk1                          0   0   0  -1  100   100  10   100  127  127   382      0      0      0        1
[448]:                   Test3  0   3   0  -1   10   100  10    53    0  119   511
cd0                             0   0   0  -1   10   100  10   100    0    0   511      0      0      0        4
hdisk0                          0   0   0  -1   10   100  10   100    0    0   511      0      0      0        4
hdisk1                          0  11   0  -1   10   100  12  -100    0  509   511      0      0     35        4
[449]:           Test3.Default  0   3   0  -1  100   100 100    53   59  119   315
cd0                             0   0   0  -1  100   100  10   100    0    0   255      0      0      0        1
hdisk0                          0   0   0  -1  100   100  10   100    0    0   255      0      0      0        1
hdisk1                          0  11   0  -1  100   100 100     4  254  376   510      0     96     24        1
[450]:            Test3.Shared  0   0   0  -1  100   100 100   100   59   59   315
cd0                             0   0   0  -1  100   100  10   100    0    0   255      0      0      0        1
hdisk0                          0   0   0  -1  100   100  10   100    0    0   255      0      0      0        1
hdisk1                          0   0   0  -1  100   100  10   100  254  254   510      0      0      0        1
[512]:                   Test4  1   0   0  -1  100   100 100   100  512  512  1023
cd0                             1   0   0  -1  100   100  10   100  512  512  1023      0      0      0        1
hdisk0                          1   0   0  -1  100   100  10   100  512  512  1023      0      0      0        1
hdisk1                          1   0   0  -1  100   100   0   100  512  512  1023      0      0      0        1
[513]:           Test4.Default  0   0   0  -1  100   100 100   100  512  512   767
cd0                             0   0   0  -1  100   100  10   100  512  512   767      0      0      0        1
hdisk0                          0   0   0  -1  100   100  10   100  512  512   767      0      0      0        1
hdisk1                          0   0   0  -1  100   100  10   100  512  512   767      0      0      0        1
[514]:            Test4.Shared  0   0   0  -1  100   100 100   100  512  512   767
cd0                             0   0   0  -1  100   100  10   100  512  512   767      0      0      0        1
hdisk0                          0   0   0  -1  100   100  10   100  512  512   767      0      0      0        1
hdisk1                          0   0   0  -1  100   100  10   100  512  512   767      0      0      0        1
```

# bqueue subcommand

## Purpose

The **bqueue** subcommand displays a queue of delayed Workload Manager I/O requests.

## Syntax

**bqueue** *effectiveaddress*

## Parameters

- *effectiveaddress* – Specifies the address of the head of the queue. This address can be obtained from the **delay** field from the output of the **bdev** subcommand.

## Aliases

**wlm_bq**

## Example

The following is an example of how to use the **bqueue** subcommand:

```
KDB(0)> bqueue 32AEE000
BUF              urap next             time
0000000032AEE000  120 0000000032AE8A00 0xA95AE221 (tod+23 ms)
0000000032AE8A00  247 0000000000000000 0xA95AE21E (tod+20 ms)

Description of output

BUF      This is the address of the buf struct in the queue
urap     The urap (class priority) of the requesting class
next     The next buf in the queue
time     The expiration time for the request (time to flush)
```

# rules subcommand

## Purpose

The **rules** subcommand displays the currently-loaded Workload Manager (WLM) assignment rules.

## Syntax

**rules**

## Parameters

There are no parameters. The output is in the following format:

```
<address>:  <classid> ("<classname>") <uidlist> <gidlist> <filelist>
```

where `<filelist>` is in the following format:

```
(<device>.<inode>.<generation>)
```

A dash ( – ) means that the list is empty (unspecified).

## Aliases

**rule**

## Example

The following is an example of how to use the **rules** subcommand:

```
KDB(0)> rules
KERN_heap+ABA0C00:  320 ("Test1") 1220 - - - -
KERN_heap+ABA0C58:  384 ("Test2") 1219 - - - -
KERN_heap+ABA0CB0:  448 ("Test3") 1218 - - - -
KERN_heap+ABA0D08:  512 ("Test4") - 1 - - -
KERN_heap+ABA0D60:  576 ("Test5") - - (8000000A00000005.000018B4.9F1CA805) - -
KERN_heap+ABA0DC8:  256 ("System") 0 - - - -
KERN_heap+ABA0E20:  128 ("Default") - - - - -
-----------------------------------------------------------
KERN_heap+ADAF000:  515 ("Test4.sub1") - - (8000000A00000005.0000187D.9F1B9E74) - -
KERN_heap+ADAF068:  516 ("Test4.sub2") - - (8000000A00000005.00001895.9F1C9F82) - -
-----------------------------------------------------------
KERN_heap+ABA2100:  579 ("Test5.sub1") - - (8000000A00000005.0000187D.9F1B9E74) - -
KERN_heap+ABA2168:  580 ("Test5.sub2") - - (8000000A00000005.00001895.9F1C9F82) - -
-----------------------------------------------------------
```

# Appendix A. kdb Command

## Purpose

Allows examining of a system dump or a running kernel.

## Syntax

**kdb** [*flags*] [ *SystemImageFile* [ *KernelFile* [*KernelModule* ... ]]]

## Description

The **kdb** command is an interactive utility for examining an operating system image or the running kernel. The **kdb** command interprets and formats control structures in the system and provides miscellaneous functions for examining a dump.

The *SystemImageFile* parameter specifies the file that contains the system image. The value can indicate a system dump, the name of a dump device, or the **/dev/pmem** special file. The default *SystemImageFile* is **/dev/pmem**.

The *KernelFile* parameter specifies the AIX kernel that kdb will use to resolve kernel symbol definitions. A kernel file must be available. When examining a system dump it is imperative that the kernel file be the same as the kernel that was used to take the system dump. The default for the *KernelFile* is **/unix**.

The *KernelModule* parameters specify the file names of any additional kernel modules which the **kdb** command uses to resolve symbol definitions not found in the kernel file itself.

Root permissions are required for use of the **kdb** command on the active system. This is required because the special file **/dev/pmem** is used. To run the **kdb** command on the active system, type the following:

```
kdb
```

**Note:** Stack tracing of the current process on a running system does not work.

To invoke the **kdb** command on a system image file, type:

```
kdb SystemImageFile
```

When kdb starts, it looks for a **.kdbinit** file in the user's home directory and in the current working directory. If a **.kdbinit** file exists in either of these locations, kdb runs all the commands inside the file as if they were entered at the interactive kdb prompt. If a **.kdbinit** file exists in both of these locations, the file in the home directory will be processed first followed by the file in the current working directory (unless the current directory is the home directory, in which case the file is processed only once).

## Flags

**-c** *CommandFile*
: Specifies a different name for the startup script file. If this option is used, then kdb will search for the *CommandFile* parameter in the home and current directories, instead of the **.kdbinit** file.

**-cp**
: Causes kdb to print out each command in the startup script files as that command is run This may be used to aid in the debugging of **.kdbinit** files (or any other file specified with the **-c** flag). Each command will be printed with a + (plus) sign in front of it.

**-h**
: Displays a short help message in regard to command line usage and a brief listing of the available command line options.

 

| | |
|---|---|
| **-i** *HeaderFile* | Makes all of the C structures defined in the *HeaderFile* parameter available for use with the kdb **print** subcommand. This option requires a C compiler to be installed on the system. If the *HeaderFile* variable needs additional **.h** files to compile, these may have to be specified with separate **-i** options as well. |
| **-k** *Module* | Instructs kdb to use the *Module* parameter as an additional kernel module for resolving symbol definitions not found in the kernel itself. Using this option is equivalent to specifying the kernel module with the *KernelModule* parameter. |
| **-l** | Disables the inline pager (that is, the `more (^C to quit) ?` prompt) in kdb. In this case the **set scroll** subcommand in kdb has no effect, and the inline pager is always disabled regardless of the scroll setting. |
| **-m** *Image* | Instructs kdb to use the *Image* parameter as the system image file. Using this option is equivalent to specifying the system image file with the *SystemImageFile* parameter. |
| **-script** | Disables the inline pager (that is, the `more (^C to quit) ?` prompt) and disables printing of most status information when kdb starts. This option facilitates parsing of the output from the **kdb** command by scripts and other programs that act as a front end for kdb. |
| **-u** *Kernel* | Instructs kdb to use the *Kernel* as the kernel file for resolving symbol definitions. Using this option is equivalent to specifying the kernel with the *KernelFile* parameter. |
| **-v** | Displays a list of all Component Dump Tables (CDTs) in the system dump file when kdb starts. CDTs list which memory regions are actually included in the system dump. If kdb is used on a live system, this option is ignored. |
| **-w** | Examines a kernel file directly instead of a system image. All kdb subcommands which normally display memory locations from the system image file will instead read data directly from *KernelFile*. Subcommands which write memory are not available. |

## Examples

The following examples demonstrate invocation options for the **kdb** command

1. To invoke the **kdb** command with the default system image and kernel image files, type:

   ```
   kdb
   ```

   The **kdb** program returns a (0)> prompt and waits for entry of a subcommand.

2. To invoke the **kdb** command using a dump file named `/var/adm/ras/vmcore.0` and the UNIX kernel file named `/unix`, type:

   ```
   kdb /var/adm/ras/vmcore.0 /unix
   ```

   The **kdb** program returns a (0)> prompt and waits for entry of a subcommand.

## Files

| | |
|---|---|
| **/usr/sbin/kdb** | Contains the **kdb** command. |
| **/dev/pmem** | Default system image file |
| **/unix** | Default kernel file |

# Appendix B. Kernel extension example files

The section contains information about the following:



It also contains various example files.

## Loading the kernel extension

To load the **demokext** kernel extension, complete the following:

1. Run the demonstration program by typing the following:
   ```
   ./demo
   ```
   This loads the **demokext** kernel extension.

   **Note:** The default prompt at this time is the dollar sign ($)
2. Stop the demonstration program by pressing the Ctrl+Z key sequence.
3. Put the demonstration program in the background by typing the following:
   ```
   bg
   ```
4. Activate the KDB kernel debugger using the Ctrl+\ key sequence.

   A KDB command prompt should appear. The default KDB prompt is`KDB(0)>`.

## Building the demonstration programs

To build the demonstration program, complete the following:

1. Save the following files in a directory.
   - “demo.c Example File” on page 435
   - “demokext.c Example File” on page 436
   - “demo.h Example File” on page 438
   - “demokext.exp example file” on page 439
2. As the root user, run the **comp_link** script. For more information on the contents of the **comp_link** script, see “comp_link Example File” on page 439.

This script produces the following:

- An executable file named **demo**
- An executable file named **demokext**
- A list file named **demokext.lst**
- A map file named **demokext.map**

 

## Generating map and list files

Assembler listing and map files are useful tools for debugging with the KDB kernel debugger. To create the assembler list file during compilation, use the **-qlist** option. Also use the **-qsource** option to get the C source listing in the same file. To create the assembler list file with these options, type the following:

```
cc -c -DEBUG -D_KERNEL -DIBMR2 demokext.c -qsource -qlist
```

In order to obtain a map file, use the **-bmap:FileName** option for the link editor. The following example creates a map file named **demokext.map**:

```
ld -o demokext demokext.o -edemokext -bimport:/lib/syscalls.exp \
-bimport:/lib/kernex.exp -lcsys -bexport:demokext.exp -bmap:demokext.map
```

## Understanding the compiler list file

The assembler and source listing is used to correlate any C source line with the corresponding assembler lines. The following is a portion of the list file, created by the **cc** command, for the demonstration kernel extension. This information is included in the compilation listing because the **-qsource** option for the **cc** command was used. The left column is the line number in the following source code:

```
   .
   .
  63 |                case 1:  /* Increment */
  64 |                        sprintf(buf, "Before increment: j=%d demokext_j=%d\n",
  65 |                                j, demokext_j);
  66 |                        write_log(fpp, buf, &bytes_written);
  67 |                        demokext_j++;
  68 |                        j++;
  69 |                        sprintf(buf, "After increment: j=%d demokext_j=%d\n",
  70 |                                j, demokext_j);
  71 |                        write_log(fpp, buf, &bytes_written);
  72 |                        break;
   .
   .
```

The assembler listing for the corresponding C code included in the compilation listing because the **-qlist** option was used with the **cc** command is as follows:

```
 .
 .
64| 0000B0 l       80BF0030   2     L4A      gr5=j(gr31,48)
64| 0000B4 l       83C20008   1     L4A      gr30=.demokext_j(gr2,0)
64| 0000B8 l       80DE0000   2     L4A      gr6=demokext_j(gr30,0)
64| 0000BC ai      30610048   1     AI       gr3=gr1,72
64| 0000C0 ai      309F005C   1     AI       gr4=gr31,92
64| 0000C4 bl      4BFFFF3D   0     CALL     gr3=sprintf,4,buf",gr3,""5",gr4-gr6,sprintf",gr1,cr[01567]",gr0",gr4"-gr12",fp0"-fp13"
64| 0000C8 cror    4DEF7B82   1
66| 0000CC l       80610040   1     L4A      gr3=fpp(gr1,64)
66| 0000D0 ai      30810048   1     AI       gr4=gr1,72
66| 0000D4 ai      30A100AC   1     AI       gr5=gr1,172
66| 0000D8 bl      4800018D   0     CALL     gr3=write_log,3,gr3,buf",gr4,bytes_written",gr5,write_log",gr1,cr[01567]",gr0",gr4"-gr12",fp0"-fp13"
66| 0000DC cal     387E0000   2     LR       gr3=gr30
67| 0000E0 l       80830000   1     L4A      gr4=demokext_j(gr3,0)
67| 0000E4 ai      30840001   2     AI       gr4=gr4,1
67| 0000E8 st      90830000   1     ST4A     demokext_j(gr3,0)=gr4
68| 0000EC l       809F0030   1     L4A      gr4=j(gr31,48)
68| 0000F0 ai      30A40001   2     AI       gr5=gr4,1
68| 0000F4 st      90BF0030   1     ST4A     j(gr31,48)=gr5
69| 0000F8 l       80C30000   1     L4A      gr6=demokext_j(gr3,0)
69| 0000FC ai      30610048   1     AI       gr3=gr1,72
69| 000100 ai      309F0084   1     AI       gr4=gr31,132
69| 000104 bl      4BFFFEFD   0     CALL     gr3=sprintf,4,buf",gr3,""6",gr4-gr6,sprintf",gr1,cr[01567]",gr0",gr4"-gr12",fp0"-fp13"
69| 000108 cror    4DEF7B82   1
71| 00010C l       80610040   1     L4A      gr3=fpp(gr1,64)
71| 000110 ai      30810048   1     AI       gr4=gr1,72
71| 000114 ai      30A100AC   1     AI       gr5=gr1,172
71| 000118 bl      4800014D   0     CALL     gr3=write_log,3,gr3,buf",gr4,bytes_written",gr5,write_log",gr1,cr[01567]",gr0",gr4"-gr12",fp0"-fp13"
72| 00011C b       48000098   1     B        CL.8,-1
 .
 .
```

With both the assembler listing and the C source listing, the assembly instructions associated with each C statement can be found. For example, compare the following C source line at line 67 of the demonstration kernel extension

```
67 |                          demokext_j++;
```

With the following assembler instructions:

```
67| 0000E0 l        80830000   1     L4A      gr4=demokext_j(gr3,0)
67| 0000E4 ai       30840001   2     AI       gr4=gr4,1
67| 0000E8 st       90830000   1     ST4A     demokext_j(gr3,0)=gr4
```

The offsets of these instructions within the demonstration kernel extension (demokext) are 0000E0, 0000E4, and 0000E8.

## Understanding map files

The binder map file is a symbol map in address order format. Each symbol listed in the map file has a storage class (CL) and a type (TY) associated with it.

Storage classes correspond to the **XMC_***TY* variables defined in the **syms.h** file. Each storage class belongs to one of the following section types:

**.text** Contains read-only data (instructions). Addresses listed in this section use the beginning of the **.text** section as origin. The **.text** section can contain one of the following storage class (CL) values:

- **DB** Debug Table. Identifies a class of sections that has the same characteristics as read only data.
- **GL** Glue Code. Identifies a section that has the same characteristics as a program code. This type of section has code to interface with a routine in another module. Part of the interface code requirement is to maintain the table of contents data structure (TOC) addressability across the call.
- **PR** Program Code. Identifies the sections that provide executable instructions for the module.
- **R0** Read Only Data. Identifies the sections that contain constants that are not modified while the program is running.
- **TB** Reserved for future use.
- **TI** Reserved for future use.
- **XO** Extended Operations code. Identifies a section of code that is to be treated as a pseudo-machine instruction.

**.data** Contains read-write initialized data. Addresses listed in this section use the beginning of the **.data** section as the origin. The **.data** section can contain one of the following storage class (CL) value types:

- **DS** Descriptor. Identifies a function descriptor. This information is used to describe function pointers in languages such as C and Fortran.
- **RW** Read Write Data. Identifies a section that contains data that is known to require change while the program is running.
- **SV** SVC. Identifies a section of code that is to be treated as a supervisory call.
- **T0** TOC Anchor. Used only by the predefined TOC symbol. Identifies the TOC special symbol that is used only by the TOC header.
- **TC** TOC Entry. Identifies address data that will reside in the TOC.
- **TD** TOC Data Entry. Identifies data that will reside in the TOC.
- **UA** Unclassified. Identifies data that contains data of an unknown storage class.

**.bss** Contains read-write data that is not initialized. Addresses listed in this section use the beginning of the **.data** section as origin. The **.bss** section contains one of the following storage class (CL) values:

**BS** BSS class. Identifies a section that contains data that is not initialized.

**UC** Unnamed Fortran Common. Identifies a section that contains read/write data.

Types correspond to the **XTY_**_TY_ variables defined in the **syms.h** file. The type (`TY`) can be one of the following values:

**ER** External Reference
**LD** Label Definition
**SD** Section Definition
**CM** BSS Common Definition

The following is the map file for the demonstration kernel extension. This file was created because of the *-bmap:demokext.map* option of the **ld** command.

```
 1   ADDRESS MAP FOR demokext
 2   *IE ADDRESS  LENGTH AL CL TY Sym#  NAME                      SOURCE-FILE(OBJECT) or IMPORT-FILE{SHARED-OBJECT}
 3   --- -------- ------ -- -- -- ----- ------------------------- ------------------------------------------------
 4    I                        ER S1    _system_configuration     /lib/syscalls.exp{/unix}
 5    I                        ER S2    fp_open                   /lib/kernex.exp{/unix}
 6    I                        ER S3    fp_close                  /lib/kernex.exp{/unix}
 7    I                        ER S4    fp_write                  /lib/kernex.exp{/unix}
 8    I                        ER S5    sprintf                   /lib/kernex.exp{/unix}
 9       00000000 000360  2 PR SD S6    <>                        demokext.c(demokext.o)
10       00000000           PR LD S7    .demokext
11       00000210           PR LD S8    .close_log
12       00000264           PR LD S9    .write_log
13       000002F4           PR LD S10   .open_log
14       00000360 000108  5 PR SD S11   .strcpy                   strcpy.s(/usr/lib/libcsys.a[strcpy.o])
15       00000468 000028  2 GL SD S12   <.sprintf>                glink.s(/usr/lib/glink.o)
16       00000468           GL LD S13   .sprintf
17       00000490 000028  2 GL SD S14   <.fp_close>               glink.s(/usr/lib/glink.o)
18       00000490           GL LD S15   .fp_close
19       000004C0 0000F8  5 PR SD S16   .strlen                   strlen.s(/usr/lib/libcsys.a[strlen.o])
20       000005B8 000028  2 GL SD S17   <.fp_write>               glink.s(/usr/lib/glink.o)
21       000005B8           GL LD S18   .fp_write
22       000005E0 000028  2 GL SD S19   <.fp_open>                glink.s(/usr/lib/glink.o)
23       000005E0           GL LD S20   .fp_open
24       00000000 0000F9  3 RW SD S21   <_$STATIC>                demokext.c(demokext.o)
25     E 000000FC 000004  2 RW SD S22   demokext_j                demokext.c(demokext.o)
26   *   00000100 00000C  2 DS SD S23   demokext                  demokext.c(demokext.o)
27       0000010C 000000  2 T0 SD S24   <TOC>
28       0000010C 000004  2 TC SD S25   <_$STATIC>
29       00000110 000004  2 TC SD S26   <_system_configuration>
30       00000114 000004  2 TC SD S27   <demokext_j>
31       00000118 000004  2 TC SD S28   <sprintf>
32       0000011C 000004  2 TC SD S29   <fp_close>
33       00000120 000004  2 TC SD S30   <fp_write>
34       00000124 000004  2 TC SD S31   <fp_open>
```

In the above map file, the **.data** section begins at the statement for line 24:

```
24       00000000 0000F9  3 RW SD S21   <_$STATIC>                demokext.c(demokext.o)
```

The TOC (Table Of Contents) starts at the statement for line 27:

```
27       0000010C 000000  2 T0 SD S24   <TOC>
```

## Using the comp_link script

The following topics include source code compilation examples and examples of link options used in the **comp_link** script:


- "demo.c Example File" on page 435
- "demokext.c Example File" on page 436
- "demo.h Example File" on page 438

## demo.c Example File

This topic contains an example file that is a source program file that loads, runs, and unloads a demonstration kernel extension.

```
#include <sys/types.h>
#include <sys/sysconfig.h>
#include <memory.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <strings.h>
#include "demo.h"

/* Extension loading data */
struct cfg_load cfg_load;
extern int sysconfig();
extern int errno;

#define NAME_SIZE 256
#define LIBPATH_SIZE 256

main(argc,argv)
int argc;
char *argv[];
       {
       char path[NAME_SIZE];
       char libpath[LIBPATH_SIZE];
       char buf[BUFLEN];
       struct cfg_kmod cfg_kmod;
       struct extparms extparms = {argc,argv,buf,BUFLEN};
       int option = 1;
       int status = 0;

       /*
        * Load the demo kernel extension.
        */
       memset(path, 0, sizeof(path));
       memset(libpath, 0, sizeof(libpath));
       strcpy(path, "./demokext");
       cfg_load.path = path;
       cfg_load.libpath = libpath;
       if (sysconfig(SYS_KLOAD, &cfg_load, sizeof(cfg_load)) == CONF_SUCC)
              {
              printf("Kernel extension ./demokext was succesfully loaded, kmid=%x\n",
                     cfg_load.kmid);
              }
       else
              {
              printf("Encountered errno=%d loading kernel extension %s\n",
                     errno, cfg_load.path);
              exit(1);
              }

       /*
        * Loop alterantely allocating and freeing 16K from memory.
        */
       option = 1;
       while (option != 0)
              {
              printf("\n\n");
              printf("0. Quit and unload kernel extension\n");
              printf("1. Configure kernel extension - increment counter\n");
              printf("2. Configure kernel extension - decrement counter\n");
```

```
        printf("\n");
        printf("Enter choice: ");
        scanf("%d", &option);
        switch (option)
              {
              case 0:
                     break;
              case 1:
                     bzero(buf,BUFLEN);
                     strcpy(buf,"sample string");
                     cfg_kmod.kmid = cfg_load.kmid;
                     cfg_kmod.cmd = 1;
                     cfg_kmod.mdiptr = (char *)&extparms;
                     cfg_kmod.mdilen = sizeof(extparms);
                     if (sysconfig(SYS_CFGKMOD,&cfg_kmod, sizeof(cfg_kmod))==CONF_SUCC)
                            {
                            printf("Kernel extension %s was successfully configured\n",
                                   cfg_load.path);
                            }
                     else
                            {
                            printf("errno=%d configuring kernel extension %s\n",
                                   errno, cfg_load.path);
                            }
                     break;
              case 2:
                     bzero(buf,BUFLEN);
                     strcpy(buf,"sample string");
                     cfg_kmod.kmid = cfg_load.kmid;
                     cfg_kmod.cmd = 2;
                     cfg_kmod.mdiptr = (char *)&extparms;
                     cfg_kmod.mdilen = sizeof(extparms);
                     if (sysconfig(SYS_CFGKMOD,&cfg_kmod, sizeof(cfg_kmod))==CONF_SUCC)
                            {
                            printf("Kernel extension %s was successfully configured\n",
                                   cfg_load.path);
                            }
                     else
                            {
                            printf("errno=%d configuring kernel extension %s\n",
                                   errno, cfg_load.path);
                            }
                     break;
              default:
                     printf("\nUnknown option\n");
                     break;
              }
        }


    /*
     * Unload the demo kernel extension.
     */
if (sysconfig(SYS_KULOAD, &cfg_load, sizeof(cfg_load)) == CONF_SUCC)
           {
           printf("Kernel extension %s was successfully unloaded\n", cfg_load.path);
           }
   else
           {
           printf("errno=%d unloading kernel extension %s\n", errno, cfg_load.path);
           }
   }
```

## demokext.c Example File

This topic contains an example file that contains the source used to demonstrate the kernel extension.

```
#include <sys/types.h>
#include <sys/malloc.h>
#include <sys/uio.h>
#include <sys/dump.h>
#include <sys/errno.h>
#include <sys/uprintf.h>
#include <fcntl.h>
#include "demo.h"


/* Log routine prototypes */
int open_log(char *path, struct file **fpp);
int write_log(struct file *fpp, char *buf, int *bytes_written);
int close_log(struct file *fpp);

/* Unexported symbol */
int demokext_i = 9;
/* Exported symbol */
int demokext_j = 99;

/*
 * Kernel extension entry point, called at config. time.
 *
 * input:
 *     cmd - unused (typically 1=config, 2=unconfig)
 *     uiop - points to the uio structure.
 */
int
demokext(int cmd, struct uio *uiop)
       {
       int rc;
       char *bufp;
       struct file *fpp;
       int fstat;
       char buf[100];
       int bytes_written;
       static int j = 0;

       /*
        * Open the log file.
        */
       strcpy(buf, "./demokext.log");
       fstat = open_log(buf, &fpp);
       if (fstat != 0) return(fstat);

       /*
        * Put a message out to the log file.
        */
       strcpy(buf, "demokext was called for configuration\n");
       fstat = write_log(fpp, buf, &bytes_written);
       if (fstat != 0) return(fstat);

       /*
        * Increment or decrement j and demokext_j based on
        * the input value for cmd.
        */
       {
       switch (cmd)
              {
              case 1:  /* Increment */
                     sprintf(buf, "Before increment: j=%d demokext_j=%d\n",
                            j, demokext_j);
                     write_log(fpp, buf, &bytes_written);
                     demokext_j++;
                     j++;
                     sprintf(buf, "After increment: j=%d demokext_j=%d\n",
                            j, demokext_j);
```

```
                write_log(fpp, buf, &bytes_written);
                break;

            case 2:  /* Decrement */
                sprintf(buf, "Before decrement: j=%d demokext_j=%d\n",
                        j, demokext_j);
                write_log(fpp, buf, &bytes_written);
                demokext_j--;
                j--;
                sprintf(buf, "After decrement: j=%d demokext_j=%d\n",
                        j, demokext_j);
                write_log(fpp, buf, &bytes_written);
                break;

            default:  /* Unknown command value */
                sprintf(buf, "Received unknown command of %d\n", cmd);
                write_log(fpp, buf, &bytes_written);
                break;
            }
        }

        /*
         * Close the log file.
         */
        fstat = close_log(fpp);
        if (fstat !=0 ) return(fstat);
        return(0);
}

/***************************************************
 * Routines for logging debug information:        *
 * open_log - Opens a log file                    *
 * write_log - Output a string to a log file      *
 * close_log - Close a log file                   *
 ***************************************************/
int open_log (char *path, struct file **fpp)
        {
        int rc;
        rc = fp_open(path, O_CREAT | O_APPEND | O_WRONLY,
                     S_IRUSR | S_IWUSR, 0, SYS_ADSPACE, fpp);
        return(rc);
        }

int write_log(struct file *fpp, char *buf, int *bytes_written)
        {
        int rc;
        rc = fp_write(fpp, buf, strlen(buf), 0, SYS_ADSPACE, bytes_written);
        return(rc);
        }

int close_log(struct file *fpp)
        {
        int rc;
        rc = fp_close(fpp);
        return(rc);
        }
```

## demo.h Example File

This topic contains the code for an include file that is used by the demo.c example file and the demokext.c example file.

```
#ifndef _demo
#define _demo

/*
 * Parameter structure
```

```
 */
struct extparms {
       int argc;
       char **argv;
       char *buf;  /* Message buffer */
       size_t len;  /*   length */
};

#define BUFLEN 4096  /* Test msg buffer length */

#endif /* _demo */
```

## demokext.exp example file

This topic contains the example code that is used as an export file for linking the **demokext** kernel extension.

```
#!/unix
* export value from demokext
demokext_j
```

## comp_link Example File

This topic contains an example script that can be used to build the demonstration program and the kernel extension.

```
#! /bin/ksh
# Script to build the demo executable and the demokext kernel extension.
cc -o demo demo.c
cc -c -DEBUG -D_KERNEL -DIBMR2 demokext.c -qsource -qlist
ld -o demokext demokext.o -edemokext -bimport:/lib/syscalls.exp -bimport:/lib/kernex.exp -lcsys -bexport:demokext.exp -bmap:demokext.map
```

# Unloading the demokext kernel extension

To unload the **demokext** kernel extension:

1. At the $ prompt, bring the demonstration program to the foreground by typing `fg` on the command line. At this point, the prompt changes to `./demo`.
2. Enter `0` to unload and exit, `1` to increment counters, or `2` to decrement counters. The prompt is not displayed again because it was shown prior to stopping the program and placing it in the background. For the purposes of this example, enter `0` to indicate that the kernel extension is to be unloaded and that the demonstration program is to terminate.

# Index

## Special characters



## A



## B



## C

# D

# E



# F

## G



## H



## I

## J

# N

# O



# P



# Q



# R



# S

# T

## U



## V

## W



## X



## Z

## Vos remarques sur ce document / Technical publication remark form

**Titre / Title :** Bull AIX 5L KDB kernel debugger and kdb command

**Nº Reférence / Reference Nº :** 86 A2 66EM 02

**Daté / Dated :** October 2005

ERREURS DETECTEES / ERRORS IN PUBLICATION

AMELIORATIONS SUGGEREES / SUGGESTIONS FOR IMPROVEMENT TO PUBLICATION

Vos remarques et suggestions seront examinées attentivement.
Si vous désirez une réponse écrite, veuillez indiquer ci-après votre adresse postale complète.

Your comments will be promptly investigated by qualified technical personnel and action will be taken as required.
If you require a written reply, please furnish your complete mailing address below.

NOM / NAME : ____________________ Date : __________

SOCIETE / COMPANY : ____________________

ADRESSE / ADDRESS : ____________________

Remettez cet imprimé à un responsable BULL ou envoyez-le directement à :

Please give this technical publication remark form to your BULL representative or mail to:

**BULL CEDOC**
**357 AVENUE PATTON**
**B.P.20845**
**49008 ANGERS CEDEX 01**
**FRANCE**

# Technical Publications Ordering Form

## Bon de Commande de Documents Techniques

**To order additional publications, please fill up a copy of this form and send it via mail to:**
Pour commander des documents techniques, remplissez une copie de ce formulaire et envoyez-la à :

**BULL CEDOC**
**ATTN / Mr. L. CHERUBIN**
**357 AVENUE PATTON**
**B.P.20845**
**49008 ANGERS CEDEX 01**
**FRANCE**

**Phone** / Téléphone : +33 (0) 2 41 73 63 96
**FAX** / Télécopie +33 (0) 2 41 73 60 19
**E–Mail** / Courrier Electronique : srv.Cedoc@franp.bull.fr

**Or visit our web sites at:** / Ou visitez nos sites web à:
**http://www.logistics.bull.net/cedoc**
`http://www-frec.bull.com` `http://www.bull.com`

| **CEDOC Reference #** Nº Référence CEDOC | **Qty** Qté | **CEDOC Reference #** Nº Référence CEDOC | **Qty** Qté | **CEDOC Reference #** Nº Référence CEDOC | **Qty** Qté |
|---|---|---|---|---|---|
| _ _ _ _ _ _ _ _ _ [ _ _ ] | | _ _ _ _ _ _ _ _ _ [ _ _ ] | | _ _ _ _ _ _ _ _ _ [ _ _ ] | |
| _ _ _ _ _ _ _ _ _ [ _ _ ] | | _ _ _ _ _ _ _ _ _ [ _ _ ] | | _ _ _ _ _ _ _ _ _ [ _ _ ] | |
| _ _ _ _ _ _ _ _ _ [ _ _ ] | | _ _ _ _ _ _ _ _ _ [ _ _ ] | | _ _ _ _ _ _ _ _ _ [ _ _ ] | |
| _ _ _ _ _ _ _ _ _ [ _ _ ] | | _ _ _ _ _ _ _ _ _ [ _ _ ] | | _ _ _ _ _ _ _ _ _ [ _ _ ] | |
| _ _ _ _ _ _ _ _ _ [ _ _ ] | | _ _ _ _ _ _ _ _ _ [ _ _ ] | | _ _ _ _ _ _ _ _ _ [ _ _ ] | |
| _ _ _ _ _ _ _ _ _ [ _ _ ] | | _ _ _ _ _ _ _ _ _ [ _ _ ] | | _ _ _ _ _ _ _ _ _ [ _ _ ] | |
| _ _ _ _ _ _ _ _ _ [ _ _ ] | | _ _ _ _ _ _ _ _ _ [ _ _ ] | | _ _ _ _ _ _ _ _ _ [ _ _ ] | |
| [ _ _ ] : **no revision number means latest revision** / pas de numéro de révision signifie révision la plus récente | | | | | |

NOM / NAME : ______________________________ Date : __________

SOCIETE / COMPANY : ______________________________

ADRESSE / ADDRESS : ______________________________

______________________________

PHONE / TELEPHONE : ______________ FAX : ______________

E–MAIL : ______________________________

**For Bull Subsidiaries** / Pour les Filiales Bull :
Identification: ______________________________

**For Bull Affiliated Customers** / Pour les Clients Affiliés Bull :
**Customer Code** / Code Client : ______________________________

**For Bull Internal Customers** / Pour les Clients Internes Bull :
**Budgetary Section** / Section Budgétaire : ______________________________

**For Others** / Pour les Autres :
**Please ask your Bull representative.** / Merci de demander à votre contact Bull.

**BULL CEDOC**
**357 AVENUE PATTON**
**B.P.20845**
**49008 ANGERS CEDEX 01**
**FRANCE**

ORDER REFERENCE
86 A2 66EM 02